Universidad de las Ciencias Informáticas

Facultad 6

Título: Extensión de Visual Paradigm for UML para el Desarrollo Dirigido por Modelos de aplicaciones de gestión de información.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autores:

Lianet Cabrera González
Enrique Roberto Pompa Torres

Tutores:

Ing. Armando Robert Lobo
Ing. Yoander Iñiguez Bermúdez

La Habana, junio 2012
"Año 54 de la Revolución"


"Organizations that do not understand the overwhelming importance of managing data and information as tangible assets in the new economy will not survive."
Tom Peters, 2001

DECLARACIÓN DE AUTORÍA

Declaramos que somos autores del presente trabajo de diploma y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre la misma, con carácter exclusivo.

Para que así conste, firmamos la presente a los ___ días del mes de _____ del año _____.

Lianet Cabrera González

_____

Firma del Autor

Enrique Roberto Pompa Torres

_____

Firma del Autor

Ing. Armando Robert Lobo

_____

Firma del Tutor

Ing. Yoander Iñiguez Bermúdez

_____

Firma del Tutor

DATOS DE CONTACTO

AUTORES:

Lianet Cabrera González
Universidad de las Ciencias Informáticas,
Ciudad de la Habana, Cuba
E-mail: lcabrerag@estudiantes.uci.cu

Enrique Roberto Pompa Torres
Universidad de las Ciencias Informáticas,
Ciudad de la Habana, Cuba
E-mail: erpompa@estudiantes.uci.cu

TUTORES:

Ing. Armando Robert Lobo
Universidad de las Ciencias Informáticas,
Ciudad de la Habana, Cuba
E-mail: arobert@uci.cu

Ing. Yoander Iñiguez Bermúdez
Universidad de las Ciencias Informáticas,
Ciudad de la Habana, Cuba
E-mail: yiniguez@uci.cu

AGRADECIMIENTOS

De Lianet:

Quiero agradecer, primero que todo, a mi familia por su apoyo, amor y comprensión; en especial a:

A mi mamá ***Laura*** *por ser la mejor madre del mundo, por ser incondicional, por apoyarme siempre y enseñarme a ser una buena persona. Gracias por enseñarme a ser fuerte y a luchar por alcanzar mis sueños. A ella debo todo lo que soy y mi gran sueño es que nunca me falte.*

A mi papá ***Luis*** *por estar siempre presente, darme su apoyo y sentirse orgulloso de todos mis logros; y a mi hermanita* ***Lucía*** *por demostrarme, a su corta edad, que para ser valiente no hace falta ser grande. El amor incondicional de ellos me impulsa a ser mejor cada día.*

A mis abuelitas por su dedicación y amor; y a los padres y madres que, afortunadamente, la vida me regaló a medida que fui creciendo: ***Félix****,* ***Eduardo****,* ***María****,* ***Alea*** *y* ***Yoanis****. Gracias por acogerme como una hija, por ser incondicionales y ayudarme a enfrentar los retos que la vida me impone.*

Agradezco a todas mis amistades, especialmente a mis mejores amigos por estar disponibles para mí en todo momento. ***Isandra****,* ***Wicho****,* ***Yohana****,* ***Andry*** *y* ***Ernesto Dueñas*** *me enseñaron el verdadero concepto de la Amistad.*

Doy gracias a ***Alain Alea Boffill*** *por haber sido mi confidente, mi amigo, mi guía y mi maestro durante los primeros 4 años de mi carrera; sobre todo en los momentos en que el mundo parecía derrumbarse.*

Doy gracias a todas las personas con las que conviví en el apartamento 110 103. En ellos encontré una gran familia y la satisfacción de haber vivido intensamente el mejor de los 5 años de mi carrera.

Agradezco a mis tutores ***Lobo*** *y* ***Yoander****. A Yoander por su dedicación; y a Lobo por formarnos como profesionales, enseñarnos que los grandes sueños valen la pena y demostrarnos que podemos lograr lo que nos propongamos. De igual manera, agradezco a los integrantes del tribunal de tesis.*

Agradezco a mi maravilloso dúo de tesis ***Enrique*** *por darme su apoyo, por compartir juntos los buenos y malos momentos, por no rendirse nunca y por ayudarme a crecer como profesional. Sin su presencia nada hubiera sido igual. Gracias Enrique, simplemente, por ser el mejor dúo de tesis.*

Finalmente, doy gracias a la Revolución cubana, en especial a nuestro eterno Comandante en Jefe Fidel Castro Ruz por crear la UCI y a Raúl Castro Ruz por mantenerla vigente; y en su conjunto hacer realidad mi sueño de ser Ingeniero en Ciencias Informáticas.

De Enrique:

El éxito no significa nada si no tienes con quien compartirlo, por eso quiero agradecerle a todos los que hicieron posible este resultado.

A mi mamá ***Nancy*** *gracias por permitirme conocer la vida, enseñarme a andar, por ser la voz que me impulsa y me apoya constantemente en todas las metas que me propongo. Gracias por enseñarme que aunque el trayecto parezca oscuro siempre hay una luz al final del camino. Por ti he llegado hasta aquí y gracias a ti seguiré adelante.*

A mi papá ***Pedro Pompa*** *porque desde pequeño has sido mi modelo a seguir, la voz fuerte que me enseñó que tengo que seguir hacia adelante, no importan los obstáculos y que todo está en el empeño que le pongas a las tareas.*

A mis Abuelos ***Norma y Roberto*** *por ser las dos personas que más orgullo sienten cuando hablan de mí. Gracias por aconsejarme y confiar en mí siempre.*

Al resto de mi extensa familia *que de una forma u otra han brindado su grano de arena para ayudarme a construir este sueño.*

A mi novia ***Laura*** *porque eres la chiquilla más shula y especial que conozco, porque has sido la fuerza que me ha acompañado todo este año, porque tenerte en mi vida me demostró que no hay nada imposible.*

A ***María Elba*** *por acogerme en su casa como un hijo y por su constante preocupación durante estos 5 años.*

A mis tutores ***Yoander y Lobo****, en especial a* ***Lobo*** *por confiar en nosotros para desarrollar esta investigación, que en un principio parecía una misión imposible, pero con su ayuda, apoyo y experiencia, paso a paso salieron los resultados.*

A todos mis amigos*, los que me han aguantado durante los 5 años, así como los se fueron uniendo durante el transcurso de esta travesía. Gracias a todos los que de una forma u otra han ayudado a mi formación, a todos los que me han ayudado a soportar la lejanía de la familia, a todos los que se convirtieron en mi familia aquí adentro. En especial a* ***Ramón*** *por convertirse en mi hermano incondicional durante todo este tiempo y a* ***Arleidis*** *por ser la amiga especial con la que siempre pude contar. A todo el piquete del apto 201 que han tenido que soportar que aprenda a tocar guitarra en su casa. En fin, gracias a todos por acogerme y hacerme parte de sus vidas.*

AGRADECIMIENTOS

*A mi compañera de tesis **Lianet** porque sin ti este sueño no se habría cumplido, por todos los momentos alegres y tristes que compartimos y las malas noches que pasamos trabajando; que aunque nos mirábamos con susto a inicios de curso, por todo el trabajo que había que realizar, gracias a su apoyo y esfuerzo todo salió adelante satisfactoriamente.*

DEDICATORIA

Dedico todos mis logros personales y profesionales a mis padres, a mi hermana y a mis amigos. A ellos debo todo lo que soy y su amor me impulsa a seguir hacia adelante.

Lianet Cabrera González

Dedico este trabajo de diploma a mi mamá y a mi papá porque a ellos les debo todo lo que soy; a mis abuelos porque siempre confiaron en mí y a toda mi familia que de una forma u otra siempre me brindaron su apoyo incondicional. A mi novia por formar parte de mis sueños y apoyarme siempre. A todos mis amigos por hacer de estos 5 años los mejores de mi vida.

Enrique Roberto Pompa Torres

RESUMEN

La presente investigación se enmarcó en la concepción e implementación de una extensión para la herramienta *Visual Paradigm for UML*, cuyo objetivo es permitir la construcción de Sistemas de Procesamiento de Transacciones, guiada por el Desarrollo Dirigido por Modelos. Con este fin se realizó un estudio acerca de las aplicaciones existentes que realizan funciones semejantes y fueron definidas las tecnologías y herramientas a utilizar en el desarrollo de la extensión. El proceso estuvo guiado por la metodología de desarrollo de software OpenUP, utilizándose como lenguaje de modelado UML 2.0 y lenguaje de programación Java 1.6. Finalmente, se obtuvo una extensión que brinda las funcionalidades necesarias para realizar las transformaciones entre diagramas de las etapas de análisis, diseño e implementación de forma automática, partiendo de los casos de uso CRUD que conforman la aplicación modelada, así como la generación del código fuente de la misma para las tecnologías Symfony 2.0 y Ext JS 3.4.

PALABRAS CLAVES:
Visual Paradigm for UML, extensión, Desarrollo Dirigido por Modelos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El inicio del siglo XX se caracterizó por poseer una fuerte tendencia hacia la industria de los servicios, predominando los servicios de información a finales del mismo. La creciente producción de información, debido a sus características, revolucionó la concepción que la sociedad tenía hasta ese momento y los cambios que acontecieron en el mundo informacional generaron la llamada "Era de la Información".

En el ámbito de la Administración de Empresas, la información constituye un factor clave para la toma de decisiones y para la gestión empresarial, siendo el eje conceptual sobre el que gravitan los Sistemas de Información (SI). “Los SI son un conjunto organizado de personas, procesos y recursos, incluyendo la información y sus tecnologías asociadas, que interactúan de forma dinámica, para satisfacer las necesidades informativas que posibilitan alcanzar los objetivos de una o varias organizaciones”[1].

Con el surgimiento de las computadoras y el inicio de la era digital todo el trabajo con datos se fue simplificando, a través de los SI informáticos. Estos últimos solo representan una subclase de los SI con el empleo intensivo de las Tecnologías de la Información y la Comunicación (TIC).

Es evidente que la implantación y uso de SI propicia el éxito, en cuanto a sus objetivos, de muchas empresas y organizaciones, por ello es por lo que se hacen cada vez más necesario en las mismas. En la actualidad los SI no solo constituyen soportes de los negocios, sino, además, un mecanismo de ventajas competitivas sostenibles, al permitir gestionar los activos tangibles e intangibles.

En Cuba los SI son utilizados en varias esferas de la sociedad con el fin de gestionar de forma rápida y eficiente la información que se almacena y maneja en sus empresas. Algunos ejemplos de SI en empresas cubanas son: Sistema de Información sobre Categorías Farmacológicas, Sistema de Información Geográfica, RODAS XXI, Versat Sarasola y Sistema de Información de Gobierno; este último en desarrollo en la Universidad de las Ciencias Informáticas.

La Universidad de las Ciencias Informáticas (UCI) ofrece soluciones ante la necesidad de informatizar la sociedad y desarrollar la industria cubana del software. Su modelo de producción está estructurado en Centros de Desarrollo, cada uno de ellos formado por departamentos donde se brindan servicios y desarrolla software. El Centro de Tecnologías de Gestión de Datos (DATEC) se caracteriza por desarrollar activos informáticos para la producción de software, relacionados con las tecnologías de bases de datos y el análisis de la información. En este, el departamento Integración de Soluciones se especializa en implementar aplicaciones de gestión de información, empleando el enfoque de Línea de Productos de Software para el desarrollo de Sistemas de Procesamiento de Transacciones (TPS por sus siglas en inglés). En dichas aplicaciones se puede observar como característica la existencia de un

dominio de negocio compuesto por muchas entidades de información a gestionar, así como reglas de negocio que actúan a modo de restricciones, todo lo cual se documenta en la Especificación de Requisitos del Software.

Desde la propia definición de un TPS se tiene el hecho de que una parte importante de los requisitos esté asociada y reflejada durante la modelación del sistema en lo que en ingeniería se conoce como patrón de casos de uso CRUD (por las siglas en inglés, *Create Read Update Destroy*). Estos son representados en el modelo de casos de uso a través del Lenguaje Unificado de Modelado (UML por sus siglas en inglés) y se traducen en una parte importante del esfuerzo total requerido en el desarrollo de un TPS.

Las consecuentes implicaciones metodológicas de realizar un CRUD en análisis, diseño, implementación y prueba conllevan al uso de una herramienta de Ingeniería de Software Asistida por Computadora (CASE por sus siglas en inglés). En el Departamento de Integración de Soluciones se promueve como herramienta CASE el *Visual Paradigm for UML* en su versión comunitaria.

La explosión de CRUD en los TPS desgasta al equipo de desarrollo en la constante actualización de los modelos, implementación de los cambios y pruebas posteriores. A ello se suma el factor tiempo y el factor tecnología, agravando la situación. En el primero de los casos el modelado empieza a postergarse o finalmente se abandona para prestar más atención a la implementación, en detrimento de la calidad total del producto. Por otro lado, las tecnologías utilizadas Ext JS y Symfony no están soportadas por *Visual Paradigm*, restando eficiencia al uso de la herramienta al no ser factible la generación de código a partir de los modelos, actividad que facilitaría la construcción de aplicaciones.

En función de lo antes expuesto se identifica el **problema científico**: ¿Cómo contribuir a la construcción de Sistemas de Procesamiento de Transacciones a partir de los artefactos de ingeniería en la herramienta *Visual Paradigm for UML*?

Se define como **objeto de estudio:** Ingeniería Basada en Modelos (MDE); enmarcado en el **campo de acción:** Extensión de la herramienta de modelado *Visual Paradigm for UML* utilizando el enfoque MDE.

Se persigue con ello el **objetivo general** de: Desarrollar una extensión para la herramienta *Visual Paradigm for UML* que permita la construcción de TPS guiada por el Desarrollo Dirigido por Modelos (MDD). El Objetivo General está desglosado en los **objetivos específicos:**

1. Construir el marco teórico de la investigación para el desarrollo de la extensión aplicando la Ingeniería Basada en Modelos (MDE).
2. Elaborar un perfil UML para modelar TPS.

3. Describir los mecanismos de análisis, diseño e implementación a aplicar.
4. Realizar el análisis, diseño e implementación de la extensión de *Visual Paradigm for UML*.
5. Realizar pruebas funcionales a la extensión implementada.

Para dar cumplimiento a los objetivos se precisan las siguientes **tareas de la investigación:**

- Revisión bibliográfica de las herramientas existentes y de los enfoques teóricos de MDE en el dominio de TPS.
- Definición y especificación de los estereotipos que conforman el perfil UML que será aplicado en el modelado.
- Descripción de los patrones a aplicar asociados a los mecanismos de análisis, diseño e implementación.
- Realización del análisis.
- Realización del diseño.
- Implementación de la extensión.
- Realización y documentación de las pruebas funcionales.

Para llevar a cabo las tareas se emplearán los **métodos:**

Teóricos:

Análisis y Síntesis: Este método es utilizado para, a partir de la investigación realizada acerca de las tendencias actuales relacionadas con MDE y las herramientas de desarrollo de software existentes, definir las tecnologías y metodologías que serán utilizadas y arribar a las conclusiones de la investigación.

Inducción - Deducción: Método aplicado para arribar a una propuesta específica de solución, partiendo del estudio de los principios de MDE.

Empíricos:

Observación: A través de este método es posible percibir los elementos relevantes relacionados con las transformaciones entre diagramas y la generación de código fuente a partir de modelos en UML.

Posibles resultados:

Se espera obtener como principal resultado una extensión de la herramienta CASE *Visual Paradigm for UML* para el Desarrollo Dirigido por Modelos (MDD) de Sistemas de Procesamiento de Transacciones (TPS). Tal extensión beneficia el proceso de desarrollo de software en DATEC, particularmente al departamento Integración de Soluciones, dado que a partir de un conjunto de

modelos se generaría el código de una solución, cuando antes debería ser escrito por programadores experimentados, siendo este proceso un trabajo tedioso y repetitivo. Lo anterior puede resumirse en:

1. Perfil UML para el Desarrollo Dirigido por Modelos de aplicaciones de gestión de información.
2. Extensión de *Visual Paradigm for UML* capaz de realizar transformaciones sucesivas de diagramas durante las etapas de análisis, diseño e implementación, facilitando estas actividades.
3. Generación del código fuente de una aplicación web de Procesamiento de Transacciones en correspondencia con los diagramas.

Estructura del trabajo de diploma:

Capítulo 1: Ingeniería Basada en Modelos para sistemas de información

Partiendo de una caracterización de los TPS, se analizan diferentes propuestas relacionadas con aplicaciones y marcos de trabajo orientados a su construcción. Se enfatiza en aquellas cuya concepción hace uso de modelos o de descripción del dominio del negocio. Respecto al modelado se realiza un análisis de las tendencias en MDE y MDD. Se analizan los elementos fundamentales referentes a UML y los perfiles UML como mecanismo de extensión. Con respecto al estado del arte, se han arribado a conclusiones fundamentales que guiaron al posterior desarrollo del trabajo.

Capítulo 2: Análisis y diseño de la extensión

Se definen las funcionalidades que debe cumplir el sistema implementado y se realizan las descripciones generales del funcionamiento del mismo. Se elabora una lista de requisitos funcionales y no funcionales que se deben tener en cuenta para la implementación de la extensión. Se muestra la propuesta de solución, un diagrama conceptual del dominio de la extensión, así como el diagrama conceptual del dominio del perfil UML definido. Se define la arquitectura de la extensión y se identifican actores, casos de uso del sistema y la relación existente entre ellos. Se define el diseño de clases, con el propósito de describir cómo se debe implementar el sistema y son descritas las clases más relevantes. Se aborda lo relacionado con los patrones arquitectónicos y de diseño presentes en la implementación. Se describe el proceso de transformación entre diagramas y la generación de código fuente y se elaboran los diagramas de interacción, específicamente los diagramas de secuencia.

Capítulo 3: Implementación y pruebas de la extensión

Se presenta el modelo de implementación a través del diagrama de componentes, los procedimientos que se deben seguir para la implementación de extensiones para *Visual Paradigm for UML* y los estándares de programación utilizados. Además, se le realizan pruebas de Caja Blanca y Caja Negra a la extensión mediante casos de prueba, para comprobar su correcto funcionamiento.

CAPÍTULO 1: Ingeniería Basada en Modelos para sistemas de información

En el presente capítulo, partiendo de una caracterización de los Sistemas de Procesamiento de Transacciones (TPS), se analizan diferentes propuestas relacionadas con aplicaciones y marcos de trabajo orientados a su construcción. Respecto al modelado, se realiza un análisis de las tendencias en MDE, fundamentalmente referidas a lenguajes específicos de dominio (DSL) y a MDD. Se profundiza en UML como lenguaje de modelado para la implementación de las principales ideas de MDE y en los perfiles UML como mecanismo de extensión. Se selecciona la tecnología, herramientas y lenguaje de programación a utilizar. Se arriba a conclusiones respecto al estado del arte, que guiaron al posterior desarrollo del trabajo.

1.1 Sistemas de Procesamiento de Transacciones (TPS)

El manejo y procesamiento eficiente de la información constituye un elemento fundamental para el desarrollo de las organizaciones. Esto se traduce en la obtención de productos y servicios con la calidad requerida, así como mayor competitividad de la empresa y eficiencia en la toma de decisiones.

Con este propósito, se ha incrementado la utilización de los SI en las empresas. De acuerdo con la función a la que vayan destinados o el tipo de usuario final, estos pueden incluirse en varias clasificaciones, entre las que se encuentran los TPS.

Son varios los autores que han realizado estudios acerca de los TPS y han elaborado definiciones sobre el tema. A continuación se muestran dos de los varios criterios que existen al respecto:

Un criterio considera que los TPS son sistemas de información computarizados creados para procesar grandes cantidades de datos relacionados con transacciones rutinarias de negocios, como las nóminas y los inventarios. Un TPS elimina los inconvenientes generados por la realización de transacciones operativas necesarias y reduce el tiempo que una vez fue requerido para llevarlas a cabo de manera manual, aunque los usuarios aún tienen que capturar datos en los sistemas computarizados[2].

Javier Garzás plantea que los sistemas conocidos como TPS o sistemas informáticos efectúan y registran las transacciones diarias rutinarias, necesarias para la marcha del negocio, capturan y procesan transacciones para hacerlas disponibles para la organización. ¿Cómo aportan valor al negocio? capturando transacciones de datos que serán usadas para la toma de decisiones[3].

A partir de las definiciones analizadas, fue posible redactar una definición que tuviese en cuenta los elementos significativos y que reflejase los diferentes enfoques para una visión sistémica del concepto:

Un TPS es un sistema de información capaz de automatizar el procesamiento de las transacciones dentro de una organización y de grandes cantidades de datos relacionados con las mismas. Dicho procesamiento consiste en la recolección, almacenamiento, modificación y recuperación de la

información generada por las transacciones producidas, reduciendo considerablemente el tiempo requerido para la realización de ese proceso, así como algunos costos asociados. La utilización de los TPS aporta valor a la empresa mediante la captura de transacciones de datos que son usadas para la toma de decisiones.

Los TPS están compuestos por elementos de hardware y de software. Se caracterizan por poseer tareas, recursos y metas predefinidas a nivel operacional, recibiendo como entradas las transacciones y eventos. Para el procesamiento de los mismos se llevan a cabo varios procesos, como: la clasificación, listado y actualización de los datos. Al finalizar el procesamiento se obtienen como resultados informes, listados, reportes, etc. que son posteriormente utilizados por el personal de operaciones, que constituye el usuario final.

Todo TPS debe poseer un rendimiento elevado (respuesta rápida), con una tasa de fallos baja (fiabilidad), donde todas las transacciones sean procesadas de la misma forma independientemente del usuario, cliente y hora del día (imparcialidad). El TPS debe, además, realizar las operaciones teniendo en cuenta los roles y responsabilidades establecidos por la organización (procesamiento controlado o compartimentación).

1.2 Papel de la Ingeniería Basada en Modelos en la construcción de TPS

Algunos autores consideran que MDE tiene por objeto la prestación de la lógica de negocio, cambiando el enfoque de desarrollo de software de la codificación para el modelado. En general, los dominios son analizados y desarrollados por medio de un metamodelo, es decir, un conjunto coherente de conceptos relacionados entre sí[4].

MDE es una metodología de desarrollo de software que se centra en la creación y explotación de modelos de dominio. El enfoque MDE está destinado a aumentar la productividad al máximo y la compatibilidad entre los sistemas a través de la reutilización de modelos estandarizados. Ello simplifica el proceso de diseño de software a través de modelos y patrones de diseño. MDE es de gran importancia a la hora de transferir los cambios en los procesos de negocio hacia los sistemas que implementan dichos procesos, facilitando el soporte a la evolución del software en cuanto a lógica y tecnología se refiere. Con el empleo de este paradigma se pretende que la escritura de código se realice automáticamente[5].

Luego de haber creado un diseño en forma de una serie de modelos, el desarrollador utiliza las transformaciones de modelos para refinarlos sucesivamente y finalmente convertirlos en código. El conjunto básico de principios de MDE se basa en los conceptos de sistema y modelo y las relaciones básicas de conformidad y de representación. En el empleo de MDE son combinados fundamentalmente los siguientes conceptos:

Lenguajes de dominio específico (DSL):

“Formalizan la estructura de la aplicación, el comportamiento y los requisitos dentro de un dominio particular. Estos lenguajes (DSL) son descritos usando metamodelos, los cuales definen relaciones entre elementos dentro de un dominio”[5].

Motores de transformación y generadores:

“Analizan ciertos aspectos de los modelos, después crean varios tipos de artefactos, tal como código fuente, entradas de simulación, descripciones de uso XML, o representaciones alternativas de dicho modelo”[5].

Un modelo es una colección de objetos y relaciones entre ellos, que juntos brindan una representación de algún sistema real; mientras que un metamodelo es un modelo para definir modelos. Cada metamodelo define un lenguaje de modelado de dominio específico, que presenta una solución al modelado de distintos tipos de sistemas de software.

Los DSL son lenguajes de tamaño pequeño, centrados en resolver algunos problemas claramente identificables en el ámbito de una aplicación. Esta tecnología permite a los diseñadores de software construir modelos gráficos adaptables a un determinado dominio de aplicación y generar código fuente usando diagramas como notación[6]. El uso de los DSL permite la obtención de notaciones de modelado específicas para cada tipo de sistema, las cuales están definidas formalmente por su metamodelo. Por otra parte, los motores de transformación facilitan la evolución de modelos, realizando transformaciones de unos modelos a otros dependiendo de las reglas de transformación entre metamodelos.

De manera general, la utilización de MDE aporta varios beneficios en el proceso de desarrollo de aplicaciones de gestión de información, especialmente en la implementación de TPS, por lo que se decidió aplicarlo en la presente investigación. Entre ellos se puede apreciar la simplificación del proceso de diseño, a través de la utilización de patrones de diseño, además de que promueve la comunicación entre los individuos y equipos de trabajo mediante la normalización de las terminologías. Además, aumenta la productividad y la compatibilidad entre sistemas mediante la reutilización de modelos estandarizados.

1.3 Desarrollo Dirigido por Modelos

Algunos autores han expresado que MDD constituye una aproximación para el desarrollo de sistemas de software basada en la separación entre la especificación de la estructura y funcionalidades esenciales del sistema y la implementación final, usando plataformas de implementación específicas. Con MDD se persigue elevar el nivel de abstracción en el desarrollo de software, dándole una mayor importancia al modelado conceptual y al papel de los modelos en el desarrollo de software actual[7].

Otros criterios afirman que la idea fundamental de MDD es sustituir al código de lenguajes de programación específicos por modelos. De este modo y en el contexto de este paradigma, los modelos son considerados como entidades de primera clase, permitiendo nuevas posibilidades de crear, analizar y manipular sistemas a través de diversos tipos de herramientas y de lenguajes[8].

Una especificación de MDD es la Arquitectura Dirigida por Modelos (MDA), en la cual las transformaciones entre modelos se definen a partir de una colección de reglas de transformación. Dicho proceso comienza cuando un desarrollador o modelador se focaliza en la construcción de un Modelo Independiente de Plataforma (PIM por sus siglas en inglés), con alto nivel de abstracción e independiente de cualquier tecnología de implementación. Luego elige una o más herramientas que le permitan ejecutar una transformación automática del PIM desarrollado, de acuerdo con las definiciones de transformación. Esto resulta en un Modelo Específico de Plataforma (PSM por sus siglas en inglés), que especifica el sistema en términos de código dependiendo de una tecnología específica, el cual puede ser luego transformado en código fuente. Para ello se hace necesario el uso de herramientas que automaticen totalmente la transformación PIM – PSM. En consecuencia, dado que el modelo de alto nivel se encuentra directamente relacionado con el código a generar, la demanda de completitud, corrección y consistencia de estos PIMs es mucho mayor que en el desarrollo tradicional de software[8].

La ventaja de aplicar este enfoque en la presente investigación radica en que, a través del desarrollo de aplicaciones web basándose en modelos, es posible especificar la estructura y funcionalidades del sistema independientemente de la tecnología de implementación que será utilizada. Esto permite la reutilización de dichos modelos en la implementación de otras aplicaciones, así como realizar la generación de código para diferentes tecnologías basándose en una misma estructura y un mismo conjunto de funcionalidades.

1.4 Lenguaje Unificado de Modelado 2.0

UML es el lenguaje estándar especificado por el *Object Management Group* (OMG por sus siglas en inglés) para visualizar, especificar, construir y documentar los artefactos de un sistema y además, sirve para el modelado del negocio y sistemas de software[8]. Este ofrece un estándar para describir los modelos, incluyendo aspectos conceptuales como procesos de negocio, funciones del sistema, expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

UML cuenta con un conjunto de notaciones y diagramas para modelar sistemas orientados a objetos y describe la semántica esencial de lo que estos diagramas y símbolos significan[9].

Algunas de las principales ventajas que tiene la utilización de UML en la construcción de sistemas de software son:

- Puede usarse en las diferentes etapas del ciclo de vida del desarrollo de sistemas.
- Es independiente del proceso o metodología de desarrollo y del lenguaje de implementación.
- Permite crear y modificar modelos expresivos mediante la combinación de los 13 tipos de diagramas que suministra en su versión 2.0.
- Es posible extender la funcionalidad de la notación gráfica mediante estereotipos y proveer una base formal para los diagramas.

Para aplicar la metodología MDE en la presente investigación resulta de vital importancia la utilización de UML como lenguaje de modelado. A través de UML es posible obtener modelos enriquecidos que serán posteriormente utilizados en las transformaciones automáticas entre diagramas y en la generación automática de código fuente, lo cual se realizará utilizando los principios de MDD y MDE.
UML es una buena alternativa para modelar PIMs y PSMs, usando para este último caso algún perfil (Profile) que especialice un modelo UML para representar cierta tecnología. La intención de los perfiles es brindar un mecanismo sencillo para la adaptación de un metamodelo existente con construcciones que son específicas de un dominio particular, plataforma o método.

1.5 Perfil UML

Actualmente UML es uno de los estándares más utilizados por la industria del software para realizar el modelado de sistemas, pero a pesar de todas las ventajas que ofrece no posee la capacidad de detallar todo tipo de dominio, en especial aquellos que presentan un alto nivel de detalle. Sin embargo, UML cuenta con algunos mecanismos de extensión, entre los que se encuentran los perfiles, que permiten llevar a cabo el modelado de sistemas con un nivel superior de detalle. Entre los ejemplos de perfiles UML actualmente aplicados en el desarrollo de software destacan:

- Perfil UML para el Aspecto de Notificación en Entornos Distribuidos CORBA.
- Perfil UML para la definición de componentes inteligentes.
- Perfil UML 2.0 para servicios de software.

Los perfiles UML forman parte del estándar de UML definido por la OMG para extender metamodelos de referencia existentes, con el fin de adaptarlos a un determinado dominio o plataforma. Su uso permite disponer de una terminología propia del dominio de la aplicación objetivo y definir una nueva notación para símbolos ya existentes más acorde con este. Un perfil UML permite añadir restricciones a un metamodelo además de las ya existentes e incorporar información que puede ser útil a la hora de transformar el modelo a otros metamodelos o a código fuente[10].
Los perfiles UML están compuestos por un conjunto de elementos relacionados entre sí, ellos son las clases, extensiones, perfiles, perfiles de aplicación, paquetes y estereotipos. Actualmente no existe un

estándar definido para la construcción de perfiles UML, pero algunos autores han propuesto una secuencia de pasos a tener en cuenta para ello[10]:

1. Definir el dominio de aplicación a modelar con el perfil.
2. Definir el perfil.
3. Definir cuáles son los elementos de UML del dominio que se están extendiendo sobre los que es posible aplicar un estereotipo.
4. Definir los valores etiquetados de los elementos del perfil.
5. Definir las restricciones que forman parte del perfil, a partir de las restricciones del dominio.

Para desarrollar la solución propuesta en la investigación, fue necesario definir un nuevo perfil UML, pues los existentes no se ajustan a las exigencias del departamento Integración de Soluciones de formalizar elementos en las etapas de análisis y diseño para su posterior implementación. Dicho perfil provee a *Visual Paradigm for UML* de un conjunto de funcionalidades que permiten realizar transformaciones entre diagramas de forma automática y realizar la generación del código fuente de un TPS.

1.6 Frameworks existentes para el desarrollo de TPS relacionados con MDE

Los frameworks son estructuras actualmente utilizadas para el desarrollo de aplicaciones en diversos lenguajes y plataformas debido a todos los beneficios que brindan en cuanto a reusabilidad, seguridad, estandarización y arquitectura interna. Son varios los frameworks que facilitan el desarrollo de sistemas y aplicaciones de clase empresarial. En la construcción de TPS destacan Naked Object y GeneXus.

Naked Object es un framework de código abierto basado en el lenguaje Java para desarrollar aplicaciones utilizando como enfoque el Desarrollo Dirigido por Dominio (DDD). A partir de la implementación de las clases que conforman el modelo de dominio de la aplicación, el framework representa automáticamente las clases en interfaces de usuario orientadas a objeto, ejecutándolas como una aplicación de cliente enriquecido o como una aplicación web[11].

Respecto al problema científico de la presente investigación, Naked Object presenta un inconveniente principal, por lo que no es factible hacer uso de este framework para dar solución al problema. Este consiste en que no ofrece la posibilidad de desarrollar aplicaciones web empleando las tecnologías Symfony 2.0 y Ext JS 3.4, pues está enfocado en el lenguaje Java.

Por su parte, GeneXus es un framework de desarrollo de aplicaciones, desarrollado por ARTech, que cubre todo el ciclo de vida. Se centra en la creación y manipulación de bases de datos, permitiendo realizar de forma automática un conjunto de tareas como el diseño y generación de la base de datos, programación, análisis de impacto y la propagación automática de los cambios. Además de ello,

permite la generación automática de programas para realizar conversiones, en cuanto a estructura y contenido se refiere, de una base de datos a otra nueva[12].

GeneXus presenta dos inconvenientes fundamentales respecto al problema científico de la presente investigación, por ello no resulta factible utilizar este framework para dar solución al problema identificado. El primer inconveniente radica en que está centrado en la creación, mantenimiento y manipulación de bases de datos, por lo que no se ajusta a todo el proceso de modelado y desarrollo de aplicaciones web. Por otra parte, la generación de código se realiza solamente para programas de conversión de bases de datos, usando lenguajes de programación como Java, C#, entre otros, no siendo así para aplicaciones web y tecnologías como Symfony 2.0 y ExtJS 3.4.

En el departamento Integración de Soluciones del centro DATEC, los frameworks utilizados en la construcción de aplicaciones web son Symfony y ExtJS. El primero de ellos se caracteriza por permitir separar la lógica de negocio, la lógica de servidor y la presentación de la aplicación web mediante la implementación del patrón arquitectónico Modelo-Vista-Controlador (MVC). Además, proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Symfony automatiza las tareas más comunes, permitiéndole al desarrollador centrarse por completo en los aspectos específicos de cada aplicación. Todas estas ventajas permiten que para el desarrollo de aplicaciones web no sea necesario comenzar la implementación desde cero.

Por su parte, Ext JS facilita el desarrollo de aplicaciones enriquecidas para internet (RIA, por sus siglas en inglés). Cuenta con un fuerte paradigma de programación basado en componentes, soportado por recursos para la programación orientada a objetos en Java Script. Ext JS flexibiliza el manejo de componentes de las páginas web como el DOM, peticiones AJAX, DHTML, entre otros. Este framework posee componentes de interfaz de usuario personalizables, con un correcto diseño y buena documentación. Además, es compatible con varios navegadores web, entre ellos Firefox y Google Chrome.

Las características de Symfony y Ext JS mencionadas anteriormente facilitan la implementación de un TPS. Teniendo en cuenta que estos son los frameworks utilizados en el departamento Integración de Soluciones para desarrollar aplicaciones web, se decidió hacer uso de ellos en la generación automática del código fuente de un TPS.

1.7 Metodología para el desarrollo de software

1.7.1 Open Unified Process (OpenUP).

La línea base de la arquitectura del departamento Integración de Soluciones define la utilización de la metodología ágil de desarrollo de software OpenUp. Teniendo en cuenta las políticas del departamento se decidió realizar una investigación sobre esta metodología y adoptar su uso para

guiar el desarrollo de la investigación. A continuación se describen las principales características de la misma.

OpenUP es una metodología dirigida a la gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental, apropiada para proyectos pequeños y de bajos recursos. Es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo[13]. Esta metodología posee varias iteraciones dentro del ciclo de vida del proyecto (ver Figura 1), que no superan las pocas semanas de duración, en dependencia de los acuerdos que se toman en el equipo de trabajo. Se debe tener en cuenta que cada iteración concluye obligatoriamente con una muestra concreta del producto, que necesariamente tiene que ser “demostrativa” o “explotable”, ya que es la forma que tiene la metodología de desarrollo de demostrarle el valor agregado al cliente.

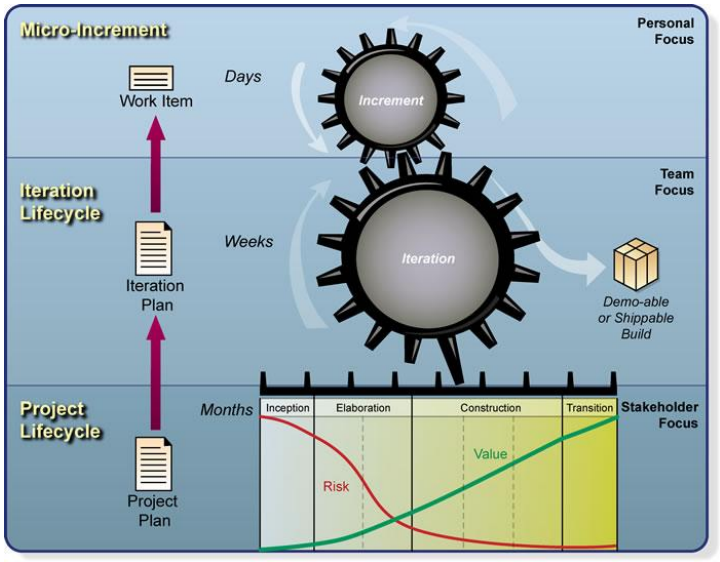


Figura 1: Ciclo de vida de OpenUp

1.8 Lenguaje de Programación

1.8.1 Lenguaje de Programación Java 1.6

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria[14].

Luego de realizar un estudio detallado acerca de los lenguajes de programación utilizados para el desarrollo de sistemas de software y teniendo en cuenta que:

- Java es el lenguaje propuesto por el API de la herramienta *Visual Paradigm for UML,* la cual fue seleccionada para el modelado de la extensión.

- Java 1.6 es el lenguaje en que se basa el motor de plantillas Apache Velocity 1.7 para realizar la generación del código fuente de aplicaciones.

Se tomó la decisión de realizar la implementación de la extensión utilizando este lenguaje.

1.9 Herramientas a utilizar

1.9.1 Herramienta de modelado Visual Paradigm for UML 8.0

Visual Paradigm for UML es una herramienta CASE que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, implementación y pruebas. Ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite construir diagramas de diversos tipos, código inverso, generar código desde diagramas y generar documentación. La herramienta también proporciona abundantes tutoriales, demostraciones interactivas y proyectos UML[15].

Algunas de las características de la herramienta que son utilizadas para el desarrollo de la extensión propuesta son:

- Integración con entornos de desarrollo: Apoyo al ciclo de vida completo de desarrollo de software en IDE como: Eclipse, NetBeans, Sun ONE, Oracle JDeveloper, JBuilder y otros.
- Detalles de casos de uso: Entorno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Multiplataforma: Soportado en la plataforma Java para varios sistemas operativos (Windows / Linux / Mac OS X).

Su versión 8.0 incluye la funcionalidad de crear y especificar perfiles UML, la cual resulta de vital importancia para la implementación y ejecución de extensiones para la herramienta. Debido a todas las características mencionadas y los beneficios que brinda para el desarrollo de software, especialmente referentes al modelado, se decidió utilizar *Visual Paradigm for UML 8.0* para el modelado de la extensión y de los TPS generados por ella. Además de ello, se tuvo en cuenta que esta constituye la herramienta que utiliza la Universidad y dentro de ella el centro DATEC para el desarrollo de software.

1.9.2 IDE de desarrollo NetBeans 7.1

El IDE NetBeans es un entorno de desarrollo integrado que permite a los desarrolladores escribir, compilar, depurar y ejecutar programas. Es un IDE de código abierto escrito completamente en Java que permite crear aplicaciones de escritorio, aplicaciones web y aplicaciones para móviles utilizando los lenguajes Java, PHP, Java Script y Ajax, entre otros. Está disponible para múltiples plataformas como son Windows, Mac, Linux y Solaris[16]. Por sus características de ser multiplataforma y compilación del lenguaje Java propuesto por el API de *Visual Paradigm for UML*, fue seleccionado

como lenguaje de programación para desarrollar la extensión. Además, se tuvo en cuenta que existe una extensión que lo integra con el motor de plantillas Apache Velocity 1.7, utilizado para la funcionalidad de generación de código fuente.

1.9.3 Motor de plantillas Apache Velocity 1.7

Apache Velocity es un motor de plantillas basado en Java que permite a los diseñadores de páginas hacer referencia a métodos definidos dentro del código Java. Los diseñadores web pueden trabajar en paralelo con los programadores Java para desarrollar sitios de acuerdo con el patrón arquitectónico MVC, permitiendo que los diseñadores se concentren únicamente en crear un sitio bien diseñado y que los programadores se encarguen solamente de escribir código de primera calidad.

Velocity separa el código Java de las páginas Web, haciendo el sitio más mantenible a largo plazo y presentando una alternativa viable a Java Server Pages (JSP) o PHP. Este motor de plantillas se puede utilizar para crear páginas web, SQL, PostScript y cualquier otro tipo de salida de plantillas. También puede ser utilizado como una aplicación independiente para generar código fuente y reportes, o como un componente integrado en otros sistemas[17].

La ventaja más importante de esta herramienta para el desarrollo de la presente investigación consiste en que posibilita efectuar la generación de código para cualquier tipo de lenguaje o tecnología. Teniendo en cuenta esta característica, se decidió utilizar Apache Velocity 1.7 como herramienta para la generación automática de código.

Conclusiones parciales

Luego de realizar un análisis sobre los principales elementos teóricos, se determinó que las metodologías MDE y MDD son las más adecuadas para ser aplicadas en el desarrollo de la solución. Se decidió implementar una extensión para la herramienta *Visual Paradigm for UML* debido a que esta herramienta permite, utilizando UML, modelar el dominio de un TPS, incluyendo en él modelos enriquecidos, a partir de los cuales serán realizadas transformaciones automáticas entre diagramas y la generación del código fuente, empleando para ello los principios de MDE y MDD.

Para la implementación de la extensión fue necesario definir un perfil UML, a través del cual serán adaptados los modelos provistos por UML al dominio específico del TPS modelado en la herramienta. Teniendo en cuenta las exigencias y necesidades del centro DATEC relacionadas con la construcción de TPS, se decidió utilizar OpenUP como metodología de desarrollo de software y como lenguaje de modelado UML 2.0, adoptado por la Universidad como estándar para el desarrollo de software, permitiendo la visualización de los artefactos del sistema.

Se emplea Java 1.6 como lenguaje de programación, propuesto por el API de desarrollo de la herramienta *Visual Paradigm for UML*. Las herramientas que serán utilizadas son: *Visual Paradigm for UML* en su versión 8.0 para el modelado de la extensión, NetBeans 7.1 como IDE de programación y Apache Velocity 1.7 como herramienta para la generación de código fuente. Para la selección de las herramientas mencionadas se tuvo en cuenta que son las propuestas por la línea base de la arquitectura del departamento Integración de Soluciones, además de las características de cada una de ellas.

CAPÍTULO 2: Análisis y diseño de la extensión

En el presente capítulo se muestra la propuesta de solución, se definen las funcionalidades que debe cumplir el sistema implementado y se realizan las descripciones generales del funcionamiento del mismo. Se elabora una lista de requisitos funcionales y no funcionales que se deben tener en cuenta para la implementación de la extensión. Se identifican actores, casos de uso del sistema, la relación existente entre ellos y el diseño de clases, con el propósito de describir cómo se debe implementar el sistema. Se define la arquitectura de la extensión y se muestra la especificación de los elementos que conforman el perfil UML definido para la misma. También se realiza una descripción de las clases más relevantes del diseño. Se aborda lo relacionado con los patrones presentes en la implementación de la extensión. Se muestra, además, el modelo del diseño y el modelo de implementación, así como los diagramas correspondientes a cada uno de ellos y los mecanismos de transformación entre los diagramas.

2.1 Propuesta de solución

Se propone realizar una extensión de la herramienta CASE *Visual Paradigm for UML* que, a partir del diagrama de casos de uso del sistema y el diagrama entidad-relación, sea capaz de realizar transformaciones sucesivas de diagramas durante las etapas de análisis, diseño e implementación. Al finalizar dichas transformaciones se obtendrá la generación del código fuente para las tecnologías Symfony 2.0 y Ext JS 3.4 y el diagrama de componentes de la aplicación web de Procesamiento de Transacciones modelada. La generación del código se hará en correspondencia con los diagramas del diseño y partiendo de la interpretación de los modelos de UML, que serán llevados a los lenguajes PHP y Java Script. El mismo tendrá aplicado un perfil UML definido a través de estereotipos que expresan la semántica del perfil.

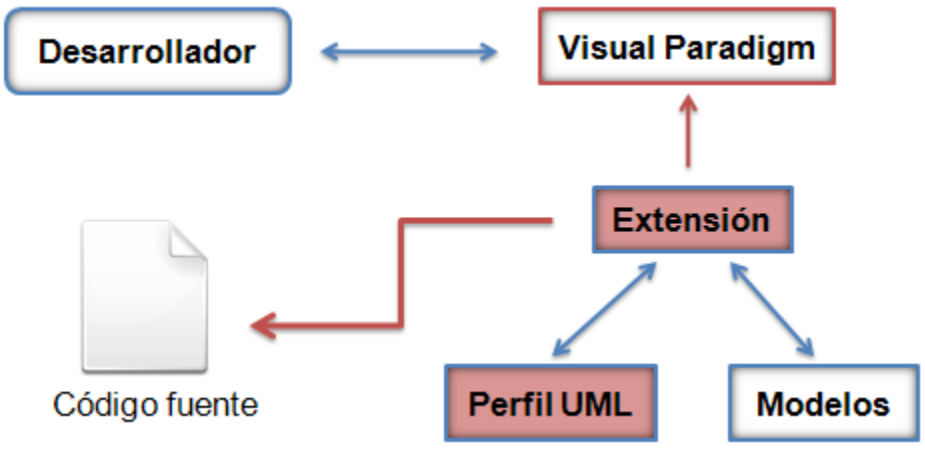


Figura 2: Propuesta de solución

2.2 Modelo de dominio de la extensión

Un modelo de dominio es un artefacto de la disciplina de análisis, construido con las reglas de UML durante la fase de concepción, que contiene conceptos propios de la realidad física. Los objetos del dominio representan las cosas que existen o los eventos que suceden en el entorno del sistema. Muchos de los objetos o clases del dominio pueden obtenerse de la especificación de requisitos. El objetivo del modelado del dominio es comprender y describir las clases más importantes dentro del contexto del sistema, por lo cual este puede ser tomado como el punto de partida para el diseño del sistema.

2.2.1 Diagrama conceptual del dominio de la extensión

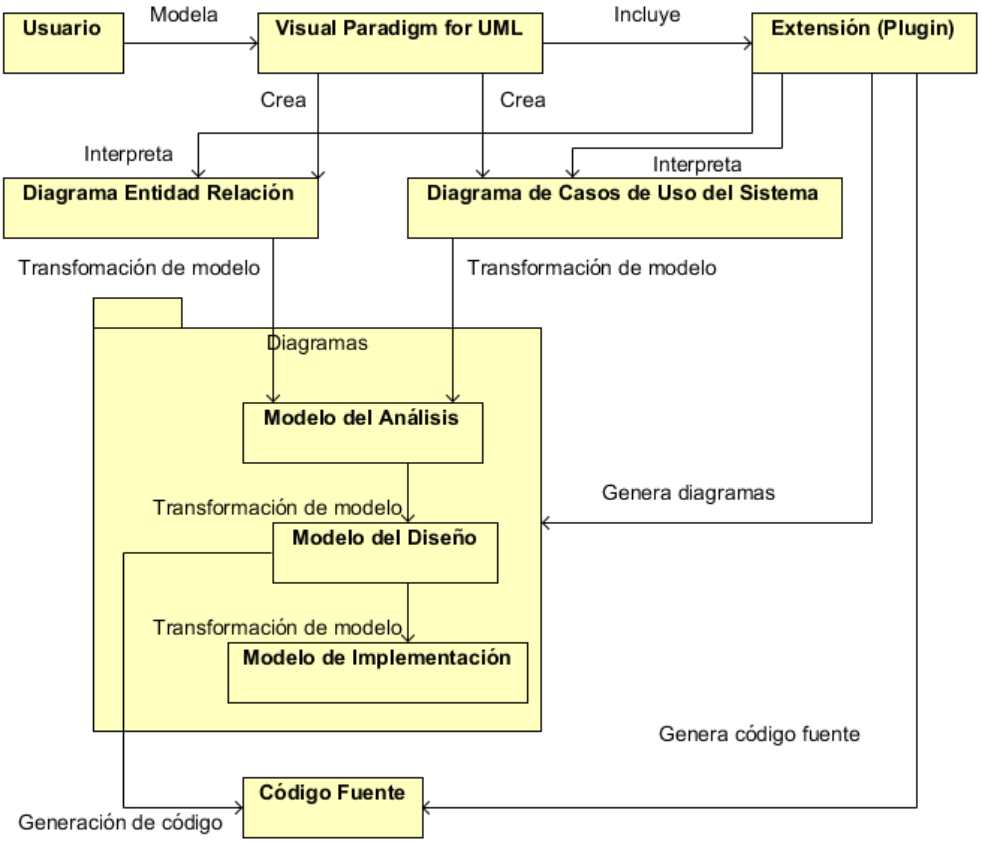


Figura 3: Modelo del dominio de la extensión

2.2.2 Definición de conceptos del modelo de dominio

- **Usuario:** El usuario es la persona responsable de modelar los diagramas en la herramienta *Visual Paradigm for UML*.

- **Visual Paradigm for UML:** Es una herramienta de modelado que permite al usuario modelar distintos tipos de diagramas, entre ellos el diagrama de casos de uso del sistema, el cual constituye el punto de partida para las funcionalidades que brinda la extensión.
- **Extensión:** Es una aplicación que al relacionarla con la herramienta *Visual Paradigm for UML* ofrece al usuario nuevas funcionalidades. En este caso permite, a partir del diagrama de casos de uso del sistema y del diagrama entidad-relación, realizar automáticamente las transformaciones entre los diagramas de las etapas de análisis, diseño e implementación y posteriormente la generación del código fuente del sistema.
- **Diagrama de casos de uso del sistema:** Es el diagrama que muestra la relación que existe entre los casos de uso del sistema y sus actores.
- **Modelo del análisis:** Es el conjunto de diagramas del análisis que permiten describir las funciones que debe desempeñar el sistema y establece una base para la creación del diseño del software.
- **Modelo del diseño:** A través del modelo del diseño se definen las clases, subsistemas e interfaces, las relaciones entre ellas y las colaboraciones que llevan a cabo los casos de uso.
- **Modelo de implementación:** Describe cómo los elementos del modelo del diseño, como por ejemplo las clases, se implementan en términos de componentes, como son los ficheros de código fuente. Describe, además, cómo se organizan los componentes y dependen unos de otros.
- **Código fuente:** Conjunto de líneas de texto generadas por la extensión, descritas utilizando el lenguaje de programación Java, que describe el funcionamiento del TPS.

2.3 Definición de un perfil UML para la extensión

Para el desarrollo de la extensión propuesta fue necesario definir un perfil UML que permite añadir restricciones a los metamodelos de UML para ajustarlos al dominio de la aplicación que será modelada en *Visual Paradigm for UML*. En el siguiente diagrama se muestran las entidades que conforman dicho perfil UML.

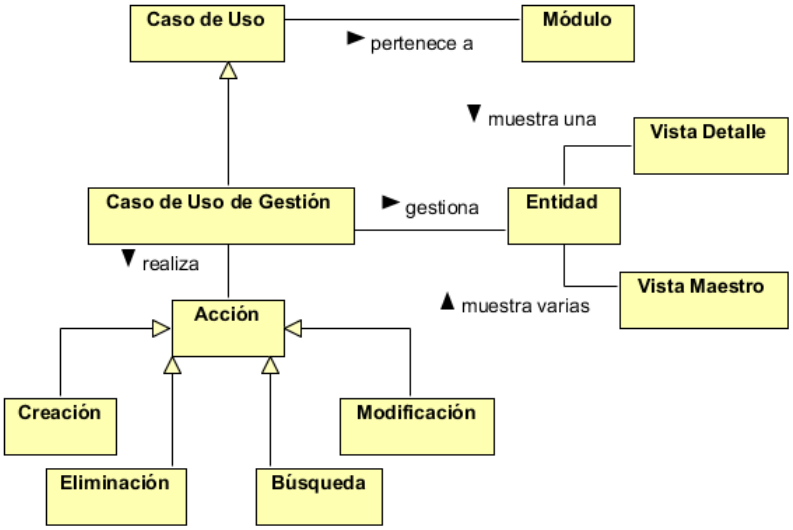


Figura 4: Modelo del dominio del perfil UML definido

En la siguiente tabla se describe cada una de las entidades que conforman el modelo del dominio del perfil UML definido.

Tabla 1: Descripción de las entidades que conforman el modelo del dominio del perfil UML definido

Nombre de la entidad	Descripción	Estereotipo	Metaclase UML 2.0 a la que extiende
Vista Detalle	La Vista Detalle generalmente es un formulario que muestra cada uno de los campos de la entidad. Es utilizada para obtener un prototipo de la Interfaz de Usuario (IU) relacionada con los datos de una entidad.	detail view	Class
Vista Maestro	La Vista Maestro es el patrón que se emplea para mostrar una colección, generalmente en forma de lista de entidades, permitiendo en muchos casos la ordenación, la búsqueda y el filtrado a través de los atributos mostrados de la entidad.	master view	Class
Acción	Son las acciones que el usuario puede realizar. Las acciones en una IU son opciones de menú o botones.	action	Method
Módulo	Es un conjunto de casos de uso relacionados funcionalmente.	module	Package

Caso de Uso	Un caso de uso es una unidad funcional y coherente del sistema, el cual representa una secuencia de interacciones entre el sistema y el actor, en función de los requisitos funcionales. El diagrama de casos de uso es el punto de partida para posteriores operaciones. Un caso de uso puede pertenecer a un módulo.	use case	
Caso de Uso de Gestión	Esta entidad es una especialización de Caso de Uso que contiene funcionalidades que permiten gestionar información perteneciente a una entidad.	crud	Use Case
Creación	Es una especialización de la entidad Acción que permite añadir en un sistema todos los datos correspondientes a una entidad.		
Eliminación	Es una especialización de la entidad Acción que posibilita eliminar entidades en un sistema.	Method	
Modificación	Es una especialización de la entidad Acción que facilita la modificación de los datos correspondientes a una entidad del sistema.	Method	
Búsqueda	Es una especialización de la entidad Acción que permite realizar la búsqueda de una entidad a partir de ciertos datos que el usuario introduzca en el sistema.	Method	
Entidad	Elemento del sistema que contiene un conjunto de propiedades. Puede ser una clase.	Method	

2.3.1 Especificación del perfil UML

La especificación del perfil UML está compuesta por un conjunto de estereotipos que heredan de sus metaclases correspondientes, los cuales son mostrados en la siguiente imagen.

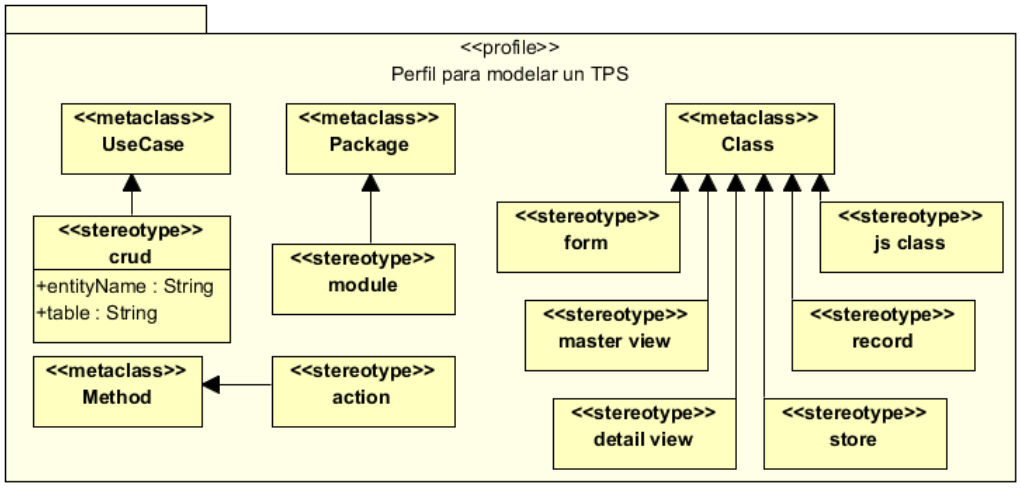


Figura 5: Especificación del perfil UML para TPS

2.3.2 Especificación de los estereotipos del perfil UML

A continuación son descritos los estereotipos que componen el perfil UML definido.

- **Estereotipo: <<detail view>>**

  Metaclase a la que extiende: Class

  Semántica del estereotipo: El estereotipo es aplicado a clases <<boundary>> en un diagrama de clases del análisis. Como las clases boundary se asocian a la interfaz de usuario, la aplicación del estereotipo guarda relación respecto al prototipo de la interfaz de usuario al corresponderse con un formulario para editar la información de la entidad gestionada.

- **Estereotipo: <<master view>>**

  Metaclase a la que extiende: Class

  Semántica del estereotipo: Debe ser aplicado a clases <<boundary>> en un diagrama de clases del análisis, por lo que al realizar el prototipo se debe cumplir con las pautas de este patrón para esta clase en particular.

- **Estereotipo: <<action>>**

  Metaclase a la que extiende: Class

  Semántica del estereotipo: Debe ser aplicado a clases <<boundary>> en un diagrama de clases del análisis, por lo que al realizar el prototipo se debe cumplir con las pautas de este patrón para esta clase en particular.

- **Estereotipo: <<module>>**

  Metaclase a la que extiende: Package

  Semántica del estereotipo: El estereotipo es aplicable a paquetes que agrupan a casos de uso dentro del marco de un sistema. Un módulo es una agrupación lógica de unidades funcionales

estrechamente relacionadas que, en su conjunto, encierran un mayor valor agregado para el usuario del sistema.

- **Estereotipo: <<crud>>**

  Metaclase a la que extiende: Use Case

  Semántica del estereotipo: Este estereotipo es aplicado a casos de uso de gestión, donde se realizan las acciones típicas de creación, modificación, búsqueda y eliminación. Incluye las etiquetas *entityName* y *tableName*, que son utilizadas para almacenar el nombre de la entidad que será gestionada y el nombre de la tabla correspondiente a la entidad respectivamente.

- **Estereotipo: <<store>>**

  Metaclase a la que extiende: Class

  Semántica del estereotipo: Este estereotipo representa una colección de datos que se le puede asignar a un componente que tenga concebida en su definición dicha propiedad de almacenamiento de datos.

- **Estereotipo: <<record>>**

  Metaclase a la que extiende: Class

  Semántica del estereotipo: Representa un registro en la base de datos o una entidad del store.

- **Estereotipo: <<form>>**

  Metaclase a la que extiende: Class

  Semántica del estereotipo: Representa un formulario con todos los campos de la entidad y está insertado en el diálogo de la vista detalle.

- **Estereotipo: <<js class>>**

  Metaclase a la que extiende: Class

  Es necesario tener en cuenta que en Java Script no existe el concepto “clase” del mismo modo que en otros lenguajes de programación, sino que la función de una clase radica en un objeto especial llamado “función constructora”, a partir de la cual se pueden crear.

  Semántica del estereotipo: Este estereotipo representa una función constructora de Java Script orientada a componentes, con sus interfaces de comunicación definidas explícitamente a través de los servicios que esta provee.

2.4 Especificación de los requisitos de la extensión

2.4.1 Requisitos funcionales

Los requerimientos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir. Constituyen el punto de partida para identificar qué debe hacer el sistema y se mantienen invariables sin importar con qué propiedades o cualidades se relacionen[18].

Para la especificación de los requisitos funcionales se deben tener en cuenta los siguientes aspectos[19]:

- Deben ser redactados de forma que sean comprensibles para usuarios sin conocimientos técnicos avanzados de informática.
- Deben especificar el comportamiento externo del sistema y evitar establecer características de su diseño.
- Deben priorizarse, distinguiendo entre requisitos obligatorios y requisitos deseables.

Teniendo en cuenta los aspectos planteados anteriormente se identificaron los siguientes requisitos funcionales, los cuales están descritos en el artefacto Especificación de Requisitos de Software (Ver Expediente de Proyecto):

RF1: Realizar la transformación hacia el análisis de un caso de uso.
Descripción: Se lleva a cabo la realización hacia el análisis del caso de uso seleccionado.
Entrada: Nombre del caso de uso seleccionado.
Salida: Diagrama de clases del análisis y diagrama de clases detallado del análisis.
RF2: Realizar la transformación hacia el análisis de un módulo.
Descripción: Se lleva a cabo la realización hacia el análisis de cada caso de uso que forma parte del módulo seleccionado.
Entrada: Nombre del módulo seleccionado.
Salida: Diagramas de clases del análisis correspondientes a los casos de uso del módulo.
RF3: Realizar la transformación hacia el análisis del sistema.
Descripción: Se lleva a cabo la realización de cada uno de los módulos que forman parte del sistema.
Entrada: Nombre del sistema.
Salida: Diagrama de clases del análisis correspondiente a cada caso de uso perteneciente a cada módulo del sistema.
RF4: Incluir métodos “get” por cada atributo.
Descripción: Se incluye en cada clase un método get por cada atributo de la clase.
Entrada: Métodos a generar “get”.
Salida: Clases con los métodos “get” incluidos.
RF5: Incluir métodos “set” por cada atributo.
Descripción: Se incluye en cada clase un método set por cada atributo de la clase.
Entrada: Métodos a generar “set”.
Salida: Clases con los métodos “set” incluidos.

RF6: Generar diagrama de clases del diseño.

Descripción: Se genera el diagrama de clases del diseño correspondiente a cada diagrama de clases del análisis.

Entrada: Diagrama de clases del análisis.

Salida: Diagrama de clases del diseño.

RF7: Generar modelo de implementación.

Descripción: Se genera el modelo de implementación a partir del modelo del diseño.

Entrada: Modelo del diseño.

Salida: Diagrama de componentes.

RF8: Generar código fuente.

Descripción: Se genera el código fuente del TPS a partir del modelo del diseño.

Entrada: Diagramas de clases del diseño, dirección donde será almacenado el código fuente.

Salida: Código fuente del sistema.

2.4.2 Requisitos no funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener y que se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Generalmente son fundamentales en el éxito del producto y normalmente están vinculados a los requisitos funcionales del sistema. Los requisitos no funcionales deben especificarse cuantitativamente para que sea posible verificar su cumplimiento[19]. Teniendo en cuenta lo antes planteado y considerando los requisitos no funcionales de la herramienta *Visual Paradigm for UML,* puesto que es esencial para el funcionamiento de la extensión, se definieron 12 requisitos no funcionales para la extensión, los cuales están descritos en el artefacto Especificación de Requisitos de Software. (Ver Expediente de Proyecto)

2.5 Modelo de casos de uso de la extensión

El modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema. A través de él son representadas las relaciones existentes entre los actores y casos de uso del sistema, donde cada caso de uso representa una unidad discreta de interacción entre un usuario y el sistema. Un caso de uso puede "incluir" la funcionalidad de otro o "extender" a otro caso de uso con su propio comportamiento[20].

2.5.1 Actores del sistema

Un actor es un usuario del sistema, que representa un conjunto coherente de roles jugados por personas, dispositivos u otros sistemas computarizados. El actor usa un caso de uso para desempeñar

alguna porción de trabajo que es de valor para el negocio. El conjunto de casos de uso al que un actor tiene acceso define su rol global en el sistema y el alcance de su acción[20].

Tabla 2: Descripción de los actores del sistema

Actor	Descripción
Analista	Es la persona encargada de, a partir del diagrama de casos de uso del sistema y el diagrama entidad-relación previamente modelados, realizar el modelo del análisis de la aplicación de forma automática a través de las opciones provistas por la extensión.
Desarrollador	Es la persona encargada de realizar las transformaciones sucesivas de diagramas durante el diseño y la implementación y posteriormente la generación del código fuente del sistema, todo ello a través de las funcionalidades provistas por la extensión.

2.5.2 Diagrama de casos de uso del sistema

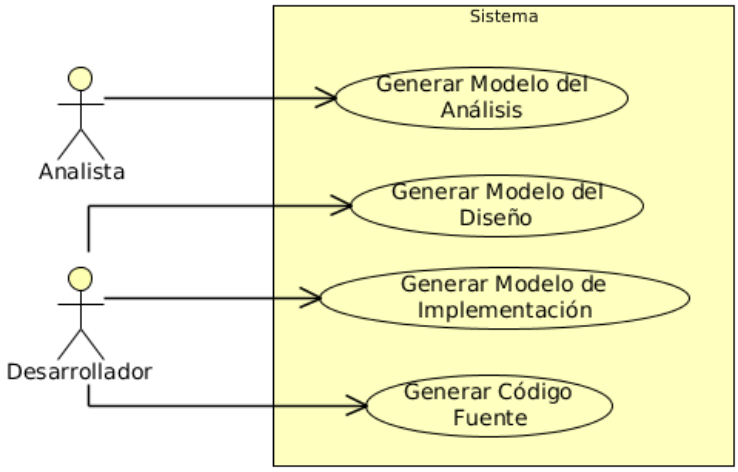


Figura 6: Diagrama de casos de uso del sistema

2.5.3 Descripción textual de los casos de uso del sistema

Luego de modelar el diagrama de casos de uso del sistema, se realizó la descripción textual de cada uno de los casos de uso que forman parte del diagrama. Dichas descripciones se encuentran reflejadas en el artefacto Modelo de Sistema (Ver Expediente de Proyecto). A continuación se muestra como ejemplo la descripción textual del caso de uso “Generar Modelo del Análisis”:

Tabla 3: Descripción textual del caso de uso "Generar Modelo del Análisis"

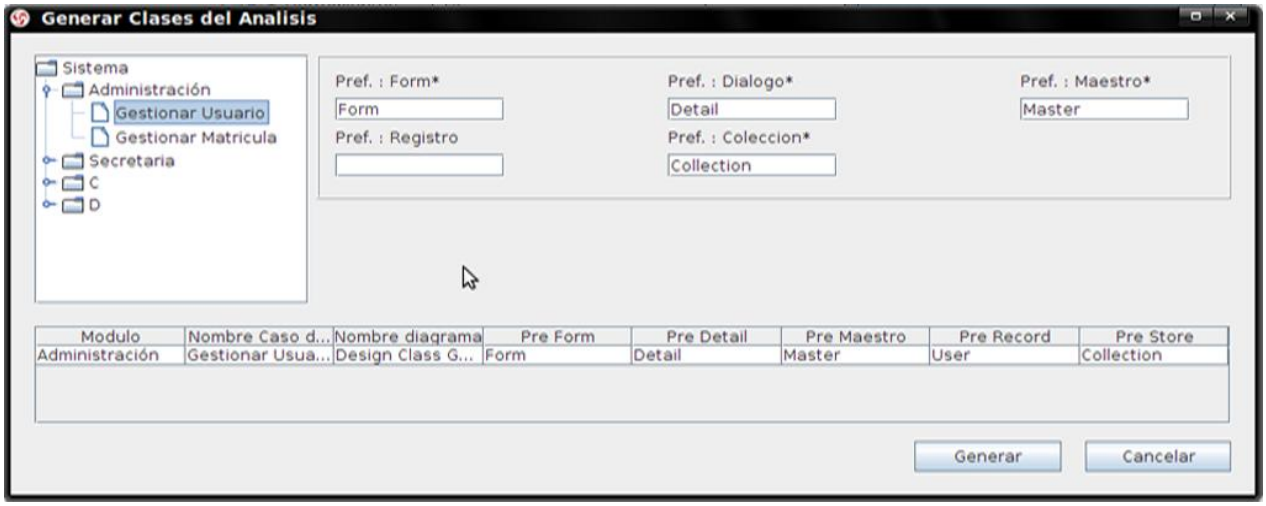
Caso de Uso:	Generar Modelo del Análisis.

Actores:	Analista
Resumen:	El caso de uso se inicia cuando el analista define los casos de uso del sistema que serán llevados al modelo del análisis, para que posteriormente se generen los diagramas de clases correspondientes. A partir del diagrama de casos de uso del sistema y del diagrama entidad-relación, el sistema debe permitir al analista generar los diagramas de clases del análisis. Para realizar los diagramas de clases del análisis existen tres variantes: realizar los diagramas del análisis de un solo caso de uso, realizar los diagramas de clases del análisis de un módulo del sistema y realizar los diagramas de clases del análisis de todos los casos de uso del sistema. Para la primera variante véase el escenario “Realizar caso de uso”, para la segunda opción véase el escenario “Realizar módulo” y para la tercera variante véase el escenario “Realizar sistema”.
Precondiciones:	Cada caso de uso CRUD contenido en el diagrama de casos de uso del sistema debe estar relacionado a través de sus valores etiquetados con la tabla del diagrama entidad-relación que le corresponde.
Referencias	**RF1, RF2, RF3 RF4, RF5.**
Prioridad	Crítico

Escenario “Realizar caso de uso”	
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El analista hace clic derecho desde el contexto de trabajo y selecciona la opción “Generar Modelo del Análisis”.	
	2. La extensión muestra una interfaz que incluye un árbol que contiene los nombres de los módulos y casos de uso del sistema, para que el analista seleccione el caso de uso que desea generar. Además, contiene una tabla y un conjunto de campos que muestran los datos relacionados con los elementos del diagrama de casos de uso, del diagrama entidad- relación y de los diagramas de clases que serán generados. (Véase el prototipo de interfaz de usuario)
3. Selecciona el caso de uso que desea	4. El sistema resalta el caso de uso seleccionado.

generar.	
5. Presiona el botón “Generar”.	
	6. La extensión visualiza en el contexto de trabajo de la herramienta los diagramas de clases del análisis que se generan y termina el caso de uso.

Prototipo de Interfaz



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
5.1 Si presiona el botón "Cancelar" no se realiza ninguna acción y finaliza el caso de uso.	

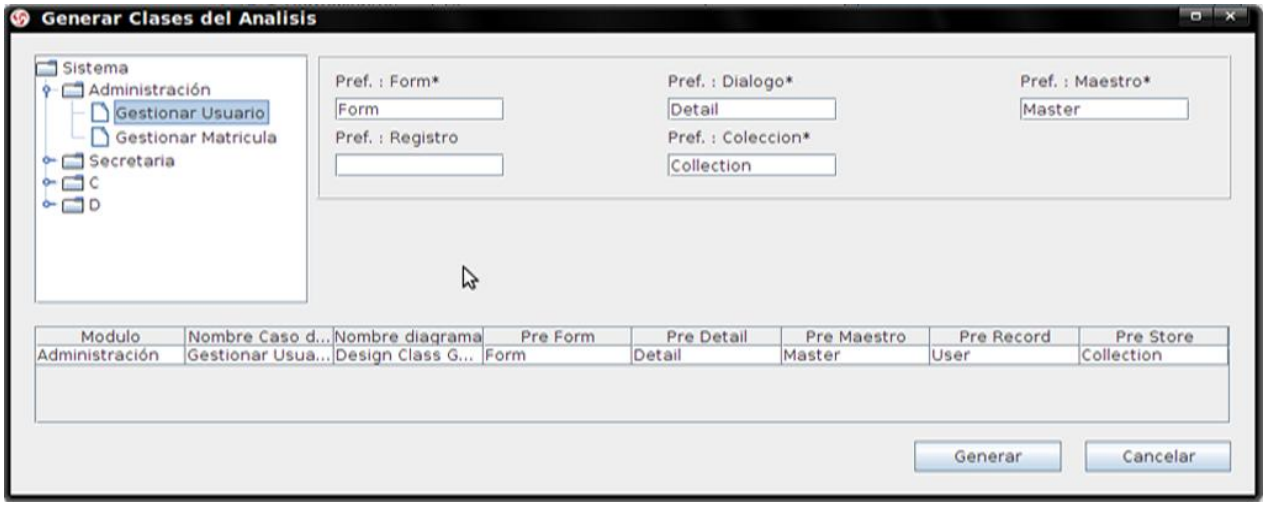
Prototipo de Interfaz

Poscondiciones	Quedó generado el modelo del análisis.

Escenario “Realizar módulo”	
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El analista hace clic derecho desde el contexto de trabajo y selecciona la opción “Generar Modelo del Análisis”.	
	2. La extensión muestra una interfaz que incluye un árbol que contiene los nombres de los módulos y casos de uso del sistema, para que el analista seleccione el módulo que desea generar. Además, contiene una tabla y un conjunto de campos que muestran los datos relacionados con los

	elementos del diagrama de casos de uso, del diagrama entidad-relación y de los diagramas de clases que serán generados. (Véase el prototipo de interfaz de usuario)
3. Selecciona el módulo que desea generar.	4. El sistema resalta el módulo seleccionado.
5. Presiona el botón “Generar”.	
	6. La extensión visualiza en el contexto de trabajo de la herramienta los diagramas de clases del análisis que se generan y termina el caso de uso.

Prototipo de Interfaz



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
5.1 Si presiona el botón "Cancelar" no se realiza ninguna acción y finaliza el caso de uso.	

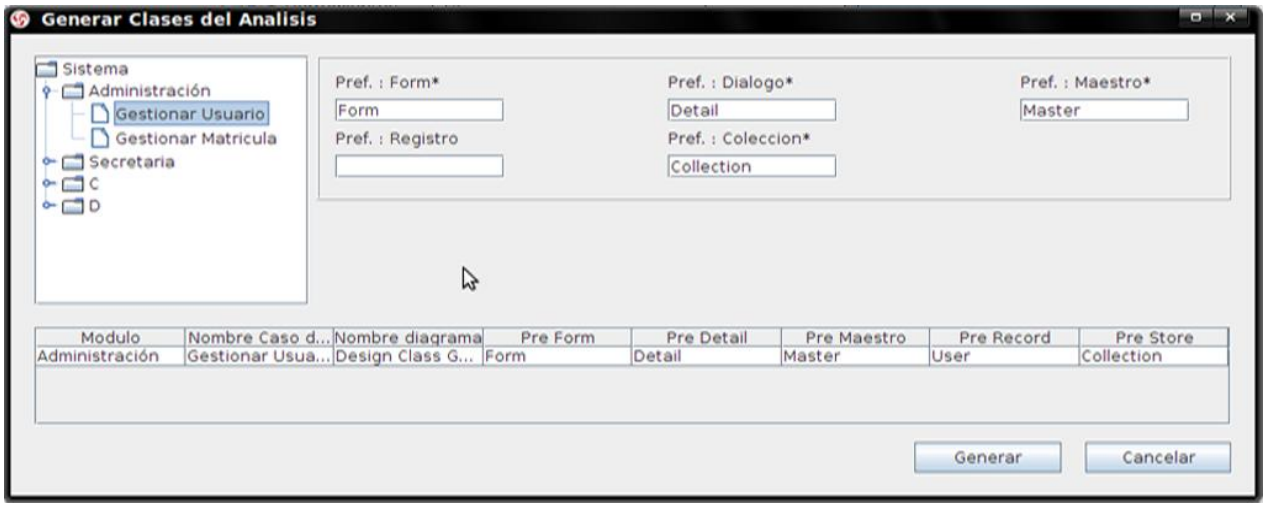
Prototipo de Interfaz

Poscondiciones	Quedó generado el modelo del análisis.

Escenario “Realizar sistema”	
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El analista hace clic derecho desde el contexto de trabajo y selecciona la opción “Generar Modelo del Análisis”.	
	2. La extensión muestra una interfaz que incluye un árbol

	que contiene los nombres de los módulos y casos de uso del sistema, para que el analista seleccione la carpeta correspondiente al sistema. Además, contiene una tabla y un conjunto de campos que muestran los datos relacionados con los elementos del diagrama de casos de uso, del diagrama entidad-relación y de los diagramas de clases que serán generados. (Véase el prototipo de interfaz de usuario)
3. Selecciona la carpeta correspondiente al sistema.	4. En la interfaz se resalta la carpeta correspondiente al sistema.
5. Presiona el botón “Generar”.	
	6. La extensión visualiza en el contexto de trabajo de la herramienta los diagramas de clases del análisis que se generan y termina el caso de uso.

Prototipo de Interfaz



Flujos Alternos

Acción del Actor	Respuesta del Sistema
5.1 Si presiona el botón "Cancelar" no se realiza ninguna acción y finaliza el caso de uso.	

Prototipo de Interfaz

Poscondiciones	Quedó generado el modelo del análisis.

2.5.4 Matriz de trazabilidad

La matriz de trazabilidad es una técnica que permite relacionar los requisitos funcionales con los diferentes elementos del desarrollo, permitiendo determinar qué requisitos quedan cubiertos por los casos de uso. A través de esta técnica es posible garantizar que todos los elementos para el desarrollo de la extensión sean ejecutados correctamente.

Tabla 4: Matriz de trazabilidad para los casos de uso

	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8
CU1	X	X	X	X	X			
CU2						X		
CU3							X	
CU4								X

2.6 Mecanismos de transformación entre diagramas en UML

Un mecanismo es una instancia de un patrón, por lo tanto, es una solución específica a un problema recurrente en un contexto único. Cualquier colaboración puede llamarse mecanismo, pero el término generalmente se reserva para colaboraciones que ofrecen una solución a un problema recurrente en aplicaciones de software, por ejemplo, para gestionar la permanencia a la que se aplica un patrón[21].

2.6.1 Mecanismo de análisis

Es un mecanismo que se utiliza en la fase inicial del proceso de diseño, durante el período de descubrimiento en que se identifican las clases y los subsistemas claves. Generalmente, los mecanismos de análisis capturan los aspectos claves de una solución independientemente de su implementación. Normalmente, no están relacionados con el dominio de problemas, sino que son conceptos informáticos. Proporcionan comportamientos específicos de la clase o componente relacionado con el dominio, o equivalen a la implementación de la cooperación entre clases o componentes[21].

La realización de un caso de uso en análisis consiste en la siguiente secuencia de pasos:

1. Elaborar un diagrama de clases del análisis, identificando las clases interfaces, controladoras y entidades.
2. Asignar responsabilidades a las clases del análisis.
3. Describir las colaboraciones entre las clases del análisis para cada escenario de peso en correspondencia con los requisitos.

4. Especificar las entidades en un modelo entidad-relación si se espera utilizar una base de datos relacional para garantizar su persistencia.

Los mecanismos del análisis aplicados en la extensión son los propuestos por la línea base de la arquitectura del departamento Integración de Soluciones.

2.6.2 Mecanismo de diseño

Es un mecanismo que se utiliza durante el proceso de diseño, durante el período en que los detalles del diseño se están decidiendo. Están relacionados con los mecanismos de análisis, de los cuales son perfeccionamientos adicionales y pueden vincular uno o más patrones de diseño y de arquitectura. No existe necesariamente ninguna diferencia en escala entre el mecanismo de análisis y el mecanismo de diseño; por lo tanto es posible hablar de un mecanismo de permanencia a nivel de análisis y a nivel de diseño y querer decir lo mismo, pero a un nivel diferente de perfeccionamiento.

Un mecanismo de diseño presupone algunos detalles del entorno de implementación, pero no está sujeto a ninguna implementación específica (como un mecanismo de implementación). Cada mecanismo de diseño tiene fortalezas y debilidades. La elección de un mecanismo de diseño específico viene determinada por las características de los objetos que lo utilizan[21].

Los mecanismos de diseño aplicados en la extensión son los propuestos por la línea base de la arquitectura del departamento Integración de Soluciones.

2.6.3 Mecanismo de implementación

Es un mecanismo utilizado durante el proceso de implementación. Son perfeccionamientos de los mecanismos de diseño, que especifican la implementación exacta del mecanismo y que utilizarán probablemente diversos patrones de implementación en su construcción. No hay necesariamente ninguna diferencia en escala entre el mecanismo de diseño y el mecanismo de implementación[21].

Los mecanismos de implementación aplicados en la extensión son los propuestos por la línea base de la arquitectura del departamento Integración de Soluciones.

2.7 Modelo del diseño de la extensión

El modelo del diseño describe la realización de casos de uso y sirve como una abstracción del modelo de aplicación y su código fuente. Es considerado un elemento esencial en la realización de las actividades de ejecución y prueba y está basado en el análisis y los requisitos de la arquitectura del sistema. A través del modelo del diseño se definen las clases, subsistemas e interfaces, las relaciones entre ellas y las colaboraciones que llevan a cabo los casos de uso[22].

2.7.1 Diagrama de clases del diseño de la extensión

Para la elaboración del modelo del diseño fueron modelados cuatro diagramas de clases del diseño, cada uno de ellos en correspondencia con un caso de uso del sistema. Dichos diagramas y sus descripciones se encuentran reflejados en el artefacto Modelo de Diseño (Ver Expediente de Proyecto).

Las clases contenidas en cada uno de los paquetes que conforman los diagramas pueden ser agrupadas teniendo en cuenta el patrón arquitectónico MVC aplicado a la extensión. El paquete "XFactory", el cual contiene a la clase *XFactoryControllerDCU*, entre otras, de acuerdo con las responsabilidades de sus clases, se corresponde con el Modelo; las clases contenidas en el paquete "actions" constituyen las clases controladoras, por lo que representan el Controlador y el paquete "dialog" se corresponde con la Vista porque contiene las interfaces de usuario mostradas por la extensión.

A continuación se muestra como ejemplo el diagrama de clases del diseño correspondiente al caso de uso "Generar Modelo del Análisis":

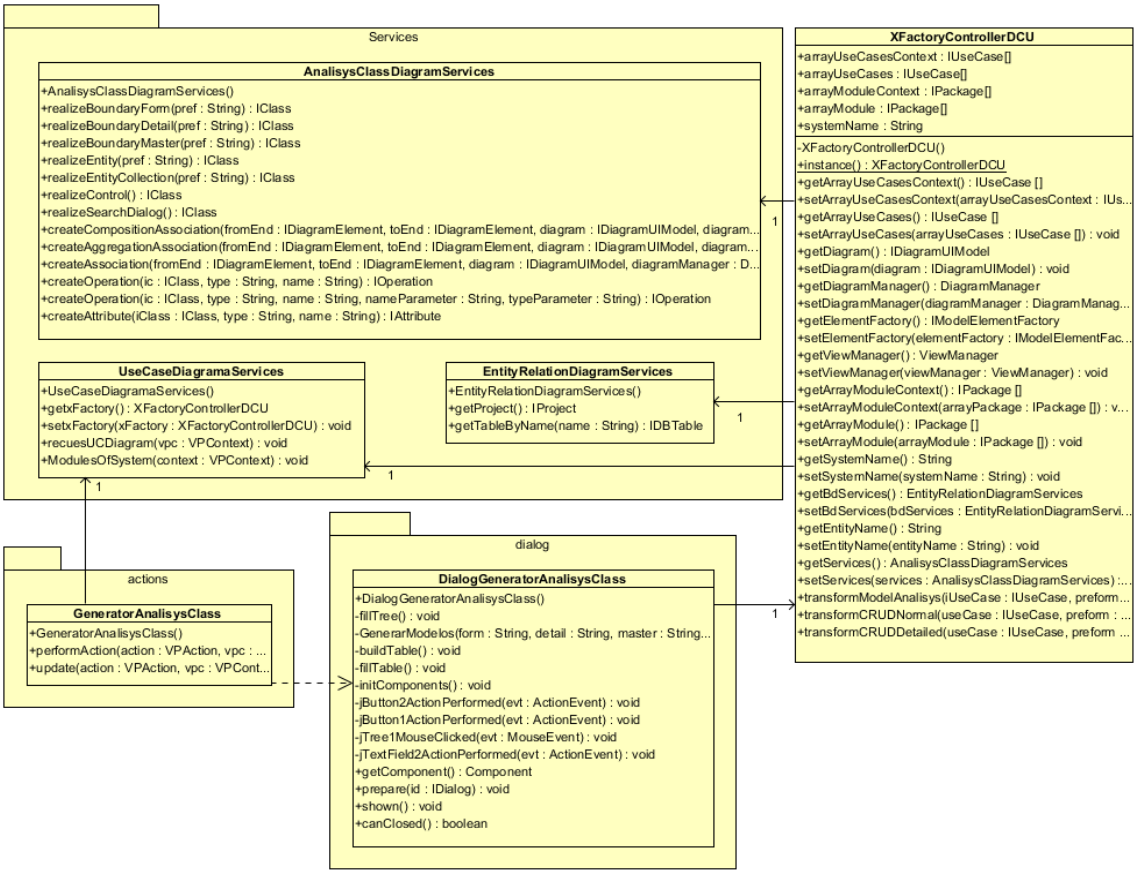


Figura 7: Diagrama de clases del diseño del caso de uso "Generar Modelo del Análisis"

2.7.2 Descripción de las clases relevantes del diseño

A partir de los diagramas de clases del diseño, se realizó la descripción de las clases relevantes que forman parte de cada uno de estos. Dichas descripciones se encuentran reflejadas en el artefacto Modelo de Diseño (Ver Expediente de Proyecto). A continuación se muestran como ejemplo las descripciones de las clases del diagrama del diseño correspondiente al caso de uso “Generar Modelo del Análisis”:

Tabla 5: Descripción de las clases más relevantes del diseño del caso de uso "Generar Modelo del Análisis"

No	Nombre de la clase	Tipo de clase	Descripción	Relación con otra clase	Tipo de relación
01	AnalisysClassDiagramServices	Entity	Esta clase contiene las funciones auxiliares que facilitan el proceso de transformación del diagrama de casos de uso al diagrama de clases del análisis.	04	Asociación
02	EntityRelationDiagramServices	Entity	Permite la obtención de las entidades del modelo del proyecto para luego relacionarlas con su respectivo caso de uso.	04	Asociación
03	UseCaseDiagramaServices	Entity	Contiene funciones auxiliares que contribuyen a capturar los elementos del contexto del diagrama de casos de uso.	04	Asociación
04	XFactoryControllerDCU	Entity	Se encarga de realizar las principales funcionalidades en el proceso de transformación del diagrama de casos de uso a diagramas de clases del análisis.		
05	GeneratorAnalisysClass	Control	Esta clase implementa las funcionalidades de la interfaz *com.vp.plugin.action.VPContextActionController,* que se encargan de	03	Asociación

			trabajar con los elementos del contexto del diagrama de casos de uso y mostrar el diálogo que permite generar las clases del análisis.		
06	DialogGeneratorAnalisysClass	Interface	Clase que muestra al analista una interfaz que reúne los campos referentes a los paquetes y casos de uso del sistema, así como la información asociada a los mismos, para que el analista seleccione los casos de uso o paquetes que desea generar.	04 05	Asociación Dependencia

2.8 Patrones utilizados.

En el desarrollo de la extensión fueron puestos en práctica un conjunto de patrones, específicamente patrones arquitectónicos, patrones de diseño y patrones de asignación de responsabilidades. Entre ellos destacan los patrones Modelo-Vista-Controlador, *Gang Of Four* (GOF por sus siglas en inglés) y *General Responsibility Assignment Software Patterns* (GRASP por sus siglas en inglés), respectivamente.

2.8.1 Patrón arquitectónico Modelo-Vista-Controlador

Visual Paradigm for UML es la herramienta CASE utilizada en el departamento Integración de Soluciones para el desarrollo de software. Esta provee una interfaz de programación de aplicaciones (API por sus siglas en inglés) que permite a los desarrolladores implementar y reutilizar clases e interfaces para desarrollar funciones agregadas de software.

Para definir la arquitectura de la extensión propuesta, resulta fundamental regirse por los elementos arquitectónicos definidos en el API de la herramienta. Esta propone una arquitectura basada en el patrón arquitectónico MVC, a través del cual es posible separar el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- Modelo: Es el responsable de administrar el comportamiento del dominio de la aplicación, empleando para ello las acciones contenidas en el *openapi.jar*. Dichas acciones están asociadas a los elementos del modelo y contemplan la creación de estereotipos, atributos, operaciones, clases y sus relaciones, entre otras.

- Vista: Es la responsable de la visualización de la información, a través de los diferentes tipos de diagramas que provee *Visual Paradigm for UML* y de las interfaces mostradas al usuario.
- Controlador: Es el responsable de interpretar los eventos producidos por el usuario, informando al modelo y/o a la vista para que cambien según resulte apropiado. Ejemplos representativos son las acciones implementadas en la extensión propuesta, las cuales capturan los elementos del modelo (Modelo) y a partir de ellos realizan acciones sobre los diagramas (Vista) y viceversa.

2.8.2 Patrones de diseño GOF

Factory Method o Método de fabricación:

El Patrón Factory Method es de tipo creación a nivel de clases. Este consiste en definir una interfaz para crear un objeto, permitiendo a las subclases decidir de qué clase instanciarlo y que una clase delegue en sus subclases la creación de objetos. Se debe utilizar cuando una clase no puede adelantar las clases de objetos que debe crear, cuando una clase pretende que sus subclases especifiquen los objetos que ella crea, cuando una clase delega su responsabilidad hacia otra clase, entre varias subclases auxiliares y se quiere tener localizada a la subclase delegada[23].

Este patrón resulta útil para la etapa de implementación de la extensión, pues garantiza la instanciación de los objetos a través de métodos creacionales, que permiten la creación de los componentes y elementos de los modelos, garantizando el correcto funcionamiento de la extensión. Su utilización se evidencia en las clases *XFactoryControllerDCp*, *XFactoryControllerDCA, DesignClassDiagramServices* y *XFactoryControllerDCU* y en esta última clase específicamente en los métodos *transformCRUDNormal* y *transformCRUDDetailed*.

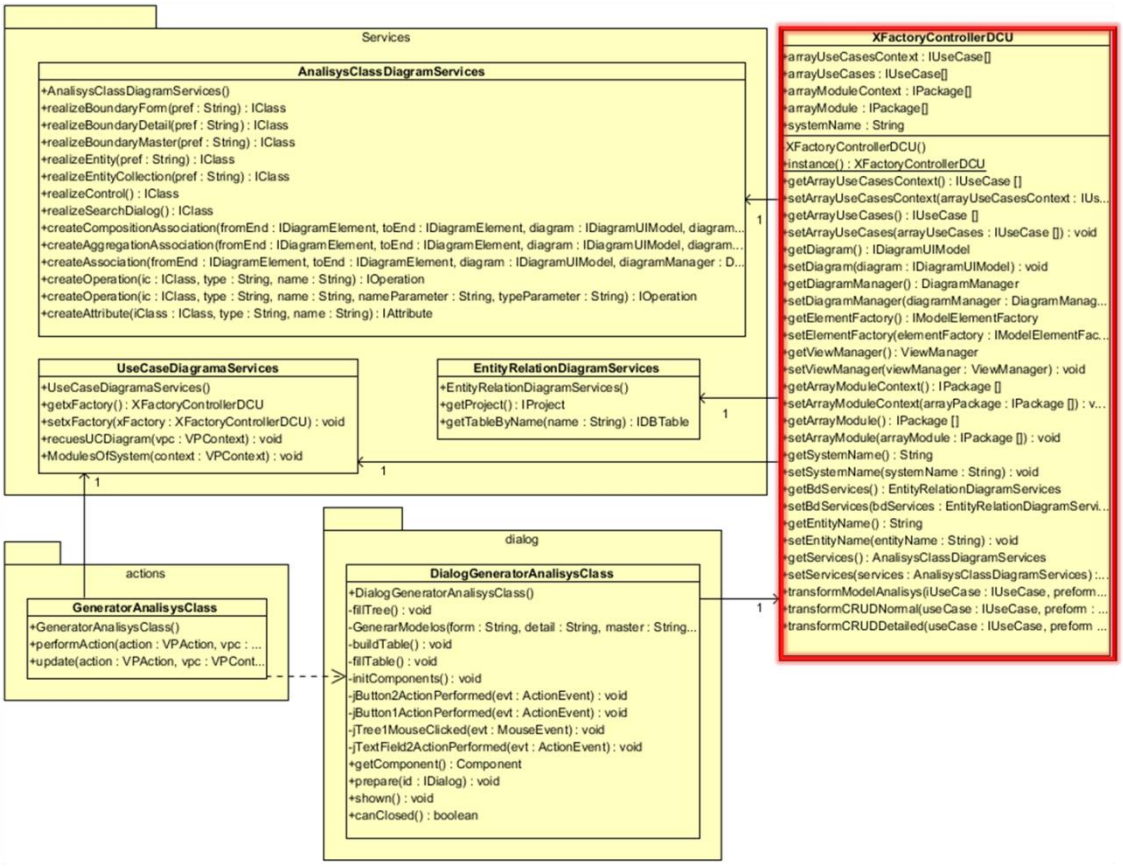


Figura 8: Ejemplo de clase donde es aplicado el patrón de Fabricación

Singleton o Solitario:

Este patrón es de tipo creacional a nivel de objetos. Su principal objetivo es garantizar que una clase solo tenga una única instancia, proporcionando un punto de acceso global a la misma. Permite realizar refinamientos en las operaciones y en la representación, mediante la especialización por herencia de "Solitario". Es fácilmente modificable para permitir más de una instancia y para controlar el número de las mismas (incluso si es variable)[23].

En la extensión para la herramienta *Visual Paradigmfor UML* es implementado este patrón en las clases *XFactoryControllerDCp*, *Util, XFactoryControllerDCD, EntityRelationDiagramServices, XFactoryControllerDCA y XFactoryControllerDCU*. En esta última clase se pone de manifiesto en el método *Util instance*, que permite la creación de una instancia de esta clase, manteniendo la consistencia entre los objetos (Ver Anexo 1).

Iterator o Iterador:

El patrón Iterador es de tipo comportamiento a nivel de objetos. A través de su utilización es posible acceder de forma secuencial a cada uno de los elementos de un objeto agregado sin exponer su representación interna. Además, permite realizar recorridos sobre objetos compuestos

independientemente de la implementación de estos. Su utilización tributa al incremento de la flexibilidad porque es posible utilizar nuevas formas de recorrer una estructura con solo modificar el iterador en uso, cambiarlo por otro o definir uno nuevo. También se facilitan el paralelismo y la concurrencia, pues es posible que dos o más iteradores recorran una misma estructura simultánea o solapadamente. Este patrón debe aplicarse cuando se necesite acceder a los elementos de un objeto agregado sin mostrar su representación interna, cuando se requiera hacer recorridos múltiples en objetos agregados y cuando se quiera proporcionar una interfaz uniforme para recorrer diferentes estructuras de agregación.

En la extensión propuesta, es aplicado este patrón en clases como *EntityRelationDiagramServices*, *UseCaseDiagramServices*, entre otras. En esta última clase se pone de manifiesto en el método *recuesUCDiagram*, que permite capturar todos los casos de uso contenidos en el diagrama de casos de uso del sistema modelado y almacenarlos en un arreglo para su posterior uso.

2.8.3 Patrones de diseño GRASP

Patrón Experto:

El uso del patrón Experto permite asignar a una clase la información necesaria para cumplir su responsabilidad. De esa forma las funcionalidades son asignadas de forma adecuada, facilitando la futura reutilización de componentes. Uno de sus beneficios consiste en permitir conservar el encapsulamiento, ya que los objetos se valen de su propia información para realizar las acciones que se les solicita[24].

Patrón Creador:

El uso del patrón Creador permite asignar a una clase la responsabilidad de crear instancias de otras[24]. Esto se evidencia dentro del marco de trabajo en las clases *AnalisysClassDiagramServices*, *DesignClassDiagramServices*, *ComponentDiagramServices* y *GeneratorSourceCode*, entre otras. En dichas clases se encuentran implementadas varias acciones dentro de las cuales se crean objetos de otras clases, lo cual evidencia que estas clases son creadoras de las que son instanciadas.

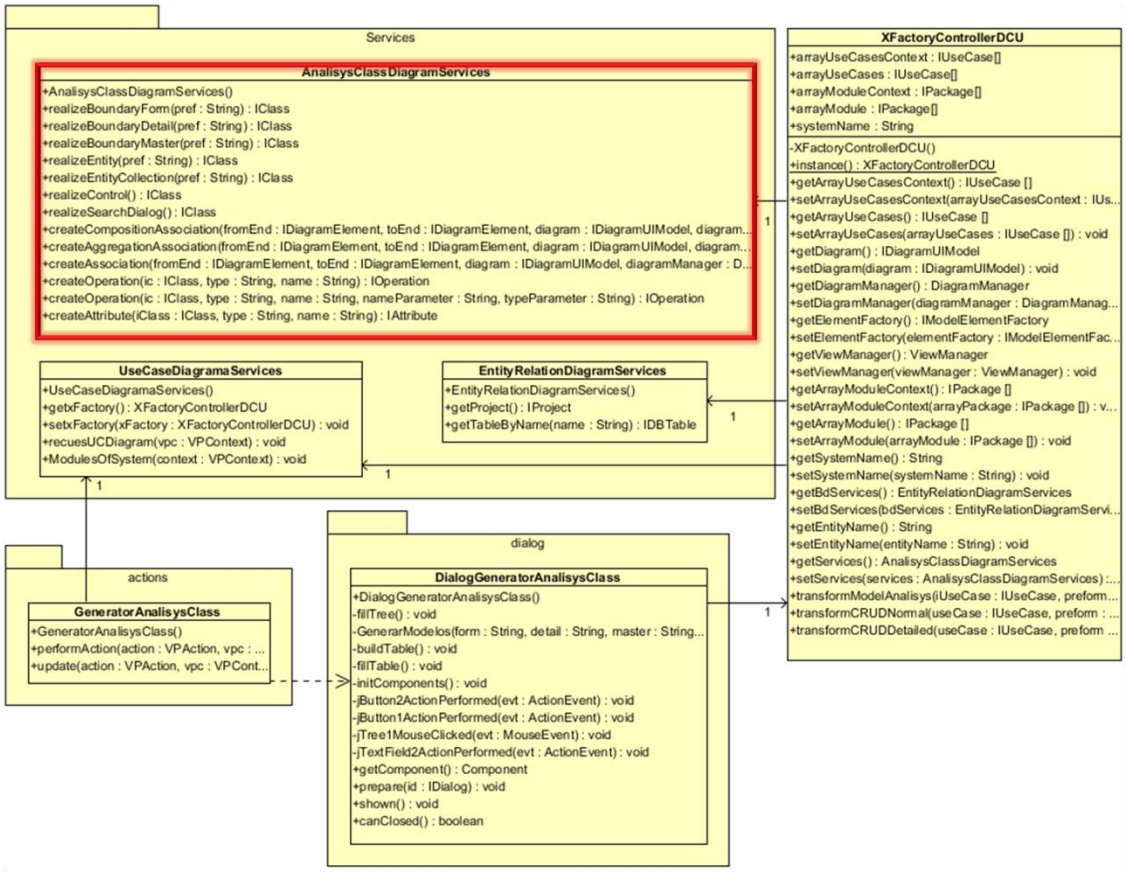


Figura 9: Ejemplo de clase donde es aplicado el patrón Creador

Patrón Alta Cohesión:

El patrón Alta Cohesión permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión[24]. Ejemplos de ello son las clases *XFactoryControllerDCU* y *UseCaseDiagramServices*, las cuales están formadas por varias funcionalidades relacionadas, siendo las responsables de definir las acciones y colaborar con otras para realizar diferentes operaciones, instanciar objetos y acceder a sus propiedades. Cada una de ellas contiene solamente operaciones correspondientes con su responsabilidad.

Patrón Bajo Acoplamiento:

La aplicación de este patrón permite asignar una responsabilidad, de modo que no se incremente el acoplamiento y, por tanto, no se produzcan los resultados negativos propios de un alto acoplamiento. Soporta el diseño de clases más independientes y reutilizables, reduciéndose el impacto de los cambios. Debe considerarse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades[24]. En la implementación de la extensión es aplicado este patrón en la

mayoría de las clases, propiciando que los componentes sean fáciles de entender por separado y de reutilizar y que no sean afectados por cambios en otros componentes.

Patrón Controlador:

La puesta en práctica de este patrón garantiza que los procesos de dominio sean manejados por la capa de los objetos del dominio y no por la de la interfaz[24]. En la extensión propuesta es aplicado este patrón en la mayoría de las clases, pues al delegar a un controlador la responsabilidad de las operaciones del sistema se favorece la reutilización de la lógica para manejar los procesos afines en aplicaciones futuras.

2.9 Descripción de las transformaciones entre diagramas en *Visual Paradigm for UML*

El proceso de transformación entre diagramas en la herramienta *Visual Paradigm for UML* se lleva a cabo partiendo de la obtención de los componentes presentes en el diagrama base.

Tabla 6: Descripción de los componentes presentes en el proceso de transformación entre diagramas del análisis, diseño e implementación

Diagrama de casos de uso	
Componentes	**Descripción**
IUse Case	Interfaz que define un caso de uso en el diagrama de casos de uso del sistema.
IDActor	Interfaz que define un actor perteneciente al diagrama de casos de uso del sistema.
IRelationShip	Interfaz que define una relación entre cualquier par de objetos de tipo IModelElement.
Diagrama entidad-relación.	
IDBTable	Interfaz que define una tabla o entidad en el diagrama entidad-relación.
IDBColumn	Interfaz que define una columna dentro de una tabla en un diagrama entidad-relación.
IDBForeignKey	Interfaz que define en una tabla la llave foránea para las relaciones entre tablas.
IRelationShip	Interfaz que define una relación entre cualquier objeto IModelElement.
Diagrama de Clases(modelo del análisis, modelo del diseño y modelo de implementación)	
IClass	Interfaz que define una clase perteneciente al diagrama de clases.
IAttribute	Interfaz que define los atributos asociados a la clase.

IOperation	Interfaz que define las operaciones asociadas a la clase.
IAssociation	Interfaz que define la relación de asociación entre clases y permite definir la agregación o composición.
IDiagramUIModel	Interfaz que permite visualizar los diagramas.
IDiagramElement	Interfaz que define los elementos de un diagrama.
ApplicationManager	Interfaz que permite el control de la aplicación.

Para llevar a cabo la transformación de un diagrama de casos de uso y un diagrama entidad-relación a un diagrama de clases del análisis, es necesario capturar todos los elementos *IDBTables* y *IUseCase* presentes en ambos diagramas. Luego se crea una instancia de la clase *VPContext* descrita en el *openapi.jar* que permite obtener el diagrama activo mediante el método *getDiagram.* Posteriormente son capturados todos los componentes, que heredan de la interfaz *IModelElement*, verificando que el componente sea un caso de uso o una tabla, de los cuales será extraída la información y convertida en atributos de las clases asociadas.

Para crear una clase es necesario crear una instancia de la interfaz *IModelElementFactory*, que permite crear una clase a través del método *createClass*. Mediante el método *setName* asociado a la interfaz *IClass* es posible asignarle un nombre a la clase. A través de la interfaz *IModelElementFactory* se crean los atributos, que son objetos de tipo *ImodelElement.* Para definir las relaciones entre clases es utilizada la interfaz *IAssociation* a través de la interfaz *IModelElementfactory*.

Una vez creadas las clases y relaciones entre ellas, es necesario visualizar el diagrama. Primero se crea una instancia de *IDiagramUIModel* mediante la interfaz *ApplicationManager,* luego se adiciona una instancia de *IDiagramElement* al diagrama, convirtiendo las clases a elementos del diagrama mediante el método *createDiagramElement* asociado a la interfaz *ApplicationManager.* Posteriormente es visualizado el *IDiagramUIModel* mediante la *ApplicationManager* a través del método *openDiagram,* mostrando en el área de trabajo de *Visual Paradigm for UML* el diagrama de clases creado a partir de la transformación del diagrama de casos de uso en diagrama de clases del análisis. Las transformaciones entre diagramas de las etapas de análisis, diseño e implementación se llevan a cabo realizando las mismas operaciones.

2.10 Descripción de la generación del código fuente en *Visual Paradigm for UML*

Para llevar a cabo la generación del código fuente de la aplicación modelada en la herramienta, es necesario capturar todos los elementos *IClass* presentes en el diagrama de clases del diseño activo. Para ello se hace una llamada al método *getDesignClass* perteneciente a la clase *SourceCodeServices*, el cual recibe por parámetros una instancia del contexto del diagrama activo. Su

responsabilidad consiste en capturar todas las clases que componen el diagrama y almacenarlas en un arreglo de tipo *IClass* que se encuentra en la clase *XFactoryControllerDCD*.

Luego se muestra al desarrollador una interfaz en la cual debe introducir los datos que se solicitan y pulsar el botón “Generar” para llevar a cabo la generación del código fuente. Esta acción activa la funcionalidad *GeneratorSource*, ubicada en la clase *XFactoryControllerDCD*, que recibe como parámetro la dirección donde se desea almacenar el código generado.

Por otra parte, la extensión cuenta con la implementación de 15 plantillas generadas a través del motor de plantillas *Apache Velocity 1.7*, cuya extensión es “.vm”. Cada una de ellas contiene en su interior el código correspondiente a una de las clases PHP o Java Script que serán generadas, así como un grupo de macros y variables que posibilitan que la escritura del código fuente se realice de forma dinámica.

Resulta imprescindible la declaración e inicialización de las variables necesarias utilizando los métodos implementados en la clase *SourceCodeService*, además de añadirlas al contexto de *Velocity*. Una vez realizadas estas acciones, se procede a la generación de las clases que conforman el código fuente, a través de los métodos *phpClassGenerator* y *jsClassGenerator*, en dependencia del tipo de clase que será generada. Estos reciben por parámetros un objeto del contexto de *Velocity*, la plantilla de *Velocity,* el objeto *IClass* correspondiente a la clase, la dirección donde se desea almacenar el código generado y la estructura de carpetas donde se almacenará la clase en cuestión. Al concluir todo el proceso mencionado, quedará generado el código fuente de la aplicación modelada en la dirección de destino especificada por el desarrollador.

2.11 Diagramas de interacción

2.11.1 Diagramas de secuencia

El diagrama de secuencia permite mostrar la secuencia del comportamiento de un caso de uso, donde cada mensaje corresponde a una operación en una clase, a un evento disparador o a una transición en una máquina de estados[25]. A través de los diagramas de secuencia, en el diseño de la extensión, es posible obtener un mejor entendimiento del sistema y detallar su comportamiento mediante objetos y mensajes enviados entre los objetos. Dichos diagramas de secuencia se encuentran reflejados en el artefacto Modelo de Diseño (Ver Expediente de Proyecto). A continuación se muestra como ejemplo el diagrama de secuencia correspondiente al caso de uso “Generar Modelo del Análisis”:

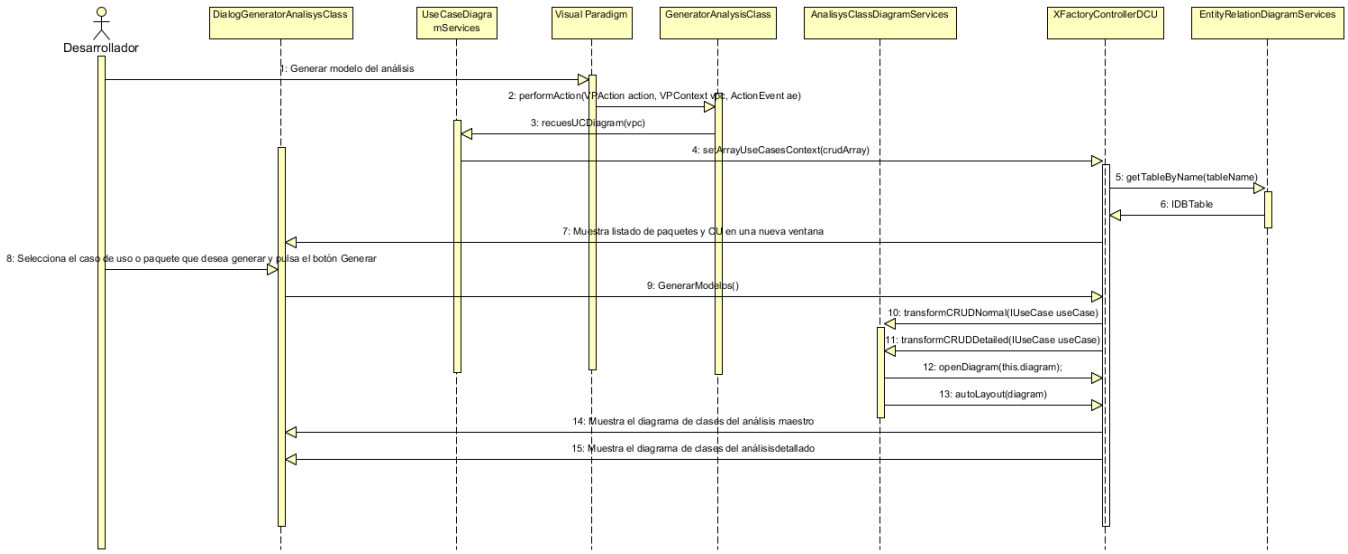


Figura 10: Diagrama de secuencia del caso de uso "Generar Modelo del Análisis"

Conclusiones parciales

La elaboración del modelo del dominio de la extensión y su descripción permitió una mayor comprensión para el desarrollo de la extensión propuesta. La identificación y especificación de los ocho requisitos funcionales, que fueron agrupados en cuatro casos de uso, así como de los requisitos no funcionales, permitió definir las funcionalidades y características que debe proveer la extensión. La descripción de los procesos de transformación entre diagramas y del proceso de generación del código fuente sirvió de guía para la correcta realización de las mencionadas actividades.

La elaboración de los diagramas de clases del diseño y los diagramas de secuencia propició una mejor comprensión de la distribución de las clases, así como de las relaciones y responsabilidades de cada una de ellas. La aplicación del patrón arquitectónico MVC y de los patrones de diseño GRASP y GOF posibilitó asignar las responsabilidades adecuadas a cada una de las clases, e independizar la lógica de la extensión de la capa de presentación, aumentando la reutilización de clases y disminuyendo el riesgo de que los cambios en la capa visual influyan en la capa lógica y viceversa. La descripción de los mecanismos de transformación del análisis, del diseño y de la implementación sirvió como guía para la correcta realización de dichas transformaciones.

CAPÍTULO 3: Implementación y pruebas de la extensión

En el presente capítulo se define el modelo de implementación que muestra el diseño de la solución, así como el diagrama de componentes de la extensión. Se describen los estándares de programación utilizados en la implementación y los procedimientos que se deben tener en cuenta para la implementación de la extensión. Se especifican, además, las pruebas realizadas a la extensión, con el objetivo de comprobar las funcionalidades de la extensión en los diferentes escenarios, para de esta forma verificar en todos los casos que los resultados de las pruebas sean los esperados.

3.1 Consideraciones sobre la implementación de extensiones a *Visual Paradigm for UML*

Para realizar la implementación de una extensión para *Visual Paradigm for UML* es necesario tener presente dos conceptos fundamentales: la estructura de desarrollo y la integración con la herramienta. Por ello, en la elaboración de la misma se deben tener en cuenta las siguientes consideraciones:

- **Definir la estructura de desarrollo:**
  A través de la librería *openapi.jar*, ubicada en "lib/openapi.jar", en el paquete de instalación de la herramienta *Visual Paradigm for UML*, es posible realizar extensiones para la misma. Una vez localizada dicha librería se debe definir la estructura de la extensión para su desarrollo. En el caso del IDE NetBeans, está conformada por un paquete que lleva el nombre "*plugin"*, en el cual están contenidos tres paquetes contenedores de clases. El primero de ellos está asociado a la configuración de la extensión, el segundo a las acciones de la misma y el tercero a los formularios o diálogos de la implementación. Antes de comenzar el desarrollo de la extensión resulta imprescindible conocer las responsabilidades de cada uno de ellos y sus relaciones con el *openapi.jar* incorporado al proyecto[26].
- **Integrar la extensión con la herramienta:**
  Luego de definir e implementar la estructura de desarrollo se debe proceder a la integración de la extensión con la herramienta. Para ello *Visual Paradigm for UML* propone una estructura de despliegue, compuesta por una carpeta con el nombre "*plugins"* creada dentro de su carpeta de instalación. Esta contiene la siguiente estructura de paquetes:

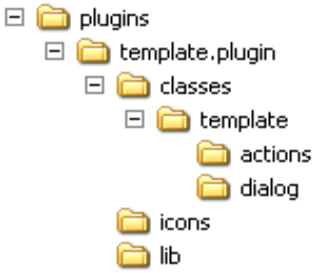


Figura 11: Estructura de despliegue de la extensión para *Visual Paradigm for UML*

La estructura de implementación de la extensión es diferente de la de integración con la herramienta, lo cual dificulta dicho proceso. Para ello es necesario utilizar la herramienta de despliegue de la extensión "*GUIVPPDeployer*". Esta cuenta con una interfaz donde el desarrollador debe insertar los valores iniciales "nombre del plugin", "dirección del proyecto plugin" y "dirección donde será desplegado"[26]. Una vez ejecutada e introducidos los datos solicitados, la extensión quedará integrada y será posible hacer uso de las nuevas funcionalidades provistas.

3.2 Modelo de implementación

El modelo de implementación describe cómo los elementos del diseño se implementan en términos de componentes, que pueden ser ficheros de código fuente, ejecutables, entre otros. Este modelo describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en los lenguajes de programación utilizados y cómo dependen los componentes unos de otros. Describe una jerarquía de subsistemas de implementación que contiene componentes e interfaces[27].

3.2.1 Diagrama de despliegue

Un diagrama de despliegue constituye una representación física de las relaciones que existen entre los componentes de hardware y software en el sistema, a través del cual es representada la configuración de los elementos de procesamiento y los componentes de software en tiempo de ejecución. Solo los componentes que son utilizados en tiempo de compilación deben mostrarse en el diagrama de despliegue.

Los componentes de la extensión son compilados por la herramienta *Visual Paradigm for UML* una vez iniciada la aplicación, formando parte de un único nodo físico de procesamiento. Por las razones antes expuestas se determinó realizar el modelo de implementación mediante el diagrama de componentes.

3.2.2 Diagrama de componentes de la extensión

Un diagrama de componentes modela la vista de implementación estática de un sistema, así como los elementos físicos que residen en un nodo, tales como ejecutables, tablas, librerías, archivos y documentos. Este está compuesto por componentes, interfaces y relaciones de dependencia, generalización, asociación y realización[28].

UML define cinco estereotipos estándar que se aplican a los componentes:

- **Executable:** Especifica un componente que se puede ejecutar en un nodo.
- **Library:** Especifica una biblioteca de objetos estática o dinámica.
- **Table:** Especifica un componente que representa una tabla de una base de datos.

- **File:** Especifica un componente que representa un documento que contiene código fuente o datos.
- **Document:** Especifica un componente que representa un documento.

A continuación se muestra el diagrama de componentes correspondiente a la extensión:

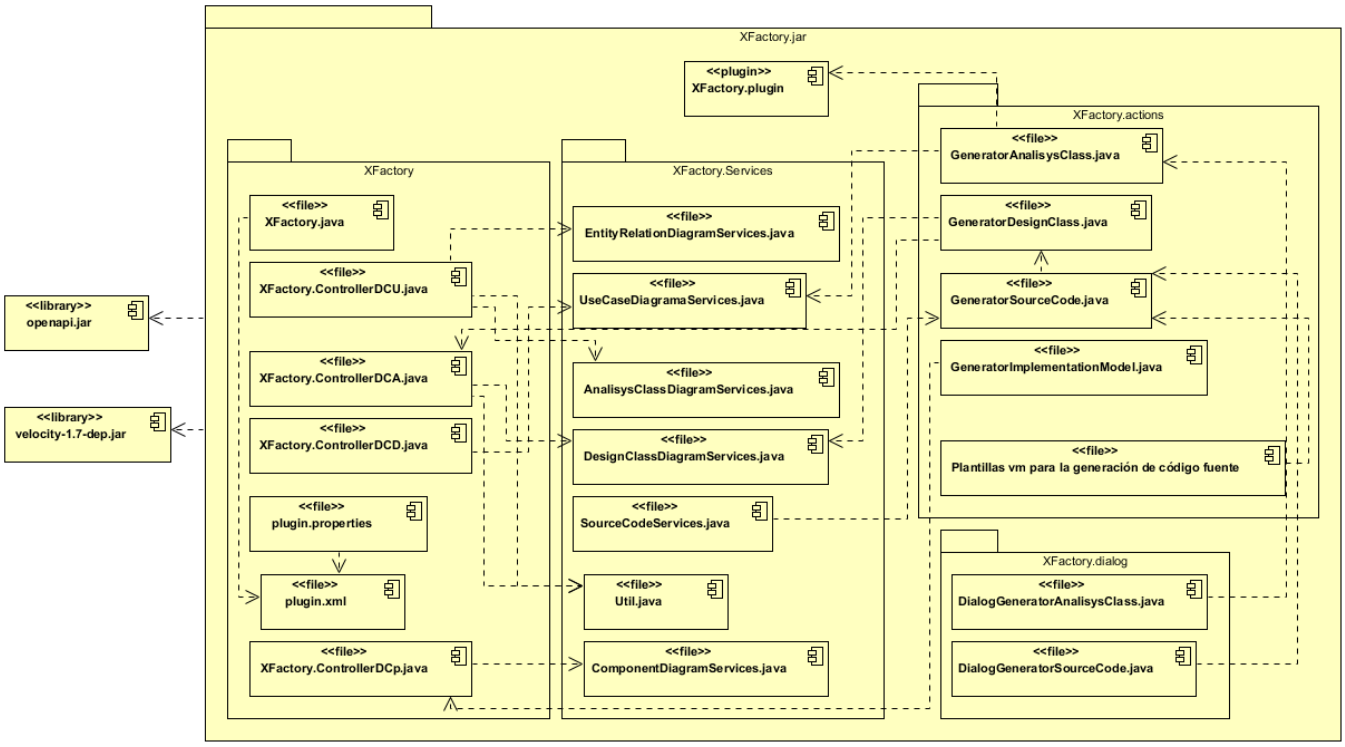


Figura 12: Diagrama de componentes de la extensión

3.2.3 Descripción de los componentes más relevantes

A continuación se describen los componentes más importantes asociados al diseño de clases propuesto para la construcción de la extensión*:*

Nombre del plugin: "*XFactory"* representa la extensión para la herramienta de modelado *Visual Paradigm for UML,* que realiza transformaciones entre diagramas y genera código fuente para las tecnologías *Symfony* y *Ext JS* con la aplicación de los principios de MDE.

Paquete XFactory.plugin: Es el paquete raíz (nombre.plugin) de la estructura de una extensión en *Visual Paradigm for UML.* Es el contenedor de paquetes asociados la configuración de la extensión, acciones y diálogos presentes en la implementación.

Paquete XFactory: Paquete que agrupa las clases asociadas a la configuración de la extensión, las cuales son: *plugin.xml*, *plugin.properties* y *Xfactory.java,* definidas en la estructura que propone la herramienta para la extensión, además de *XFactoryControllerDCU.java*, *XFactoryControllerDCA.java* y *XFactoryControllerDCD.java*.

Plugin.xml: Su función consiste en la configuración de la extensión. Define el nombre de la extensión, descripción, proveedor, librerías a utilizar y las acciones a ejecutar.

Plugin.properties: Su función es definir propiedades asociadas a los eventos y acciones de la extensión.

Xfactory.java: Su función está dirigida a la carga y descarga de la extensión mediante la clase *VPPluginInfo* que provee el *openapi.jar*, capturando la información definida en el archivo *plugin.xml*. Este componente implementa la interfaz *VPPlugin* definida en el *openapi.jar* de la misma, implementando a su vez los métodos *loaded* y *unloaded* para su función.

XFactoryControllerDCU.java: Se encarga de realizar las principales funcionalidades en el proceso de transformación del diagrama de casos de uso a diagramas de clases del análisis.

XFactoryControllerDCA.java: Se encarga de realizar las funcionalidades principales en el proceso de transformación del diagrama de clases del análisis al diagrama de clases del diseño.

XFactoryControllerDCp.java: Se encarga de realizar las principales funcionalidades en el proceso de transformación del diagrama de clases del diseño al diagrama de componentes.

XFactoryControllerDCD.java: Se encarga de realizar las principales funcionalidades en el proceso de transformación del diagrama de clases del diseño al código fuente.

Paquete Xfactory.Services: Paquete que agrupa las clases asociadas a los servicios de procesamiento de los elementos del diagrama de casos de uso del sistema y del diagrama entidad-relación capturados. Además, incluye clases que tienen como función la captura y creación de clases y las relaciones entre ellas. Este conjunto de clases es: *Util.java*, *UseCaseDiagramaServices.java*, *EntityRelationDiagramServices.java*, *AnalisysClassDiagramServices.java*, *SourceCodeServices.java y DesignClassDiagramServices.java*.

Util.java: Contiene métodos auxiliares que permiten la visualización de mensajes en la consola de *Visual Paradigm for UML*.

UseCaseDiagramaServices.java: Contiene funciones auxiliares que contribuyen a capturar los elementos del contexto del diagrama de casos de uso.

EntityRelationDiagramServices.java: Permite la obtención de las entidades del modelo del proyecto para luego relacionarlas con su respectivo caso de uso.

AnalisysClassDiagramServices.java: Esta clase contiene las funciones auxiliares que facilitan el proceso de transformación del diagrama de casos de uso al diagrama de clases del análisis.

DesignClassDiagramServices.java: Contiene las funcionalidades auxiliares que contribuyen a transformar los diagramas de clases del análisis en diagramas de clases del diseño.

SourceCodeServices.java: Clase que tiene la responsabilidad de generar las clases que conforman el código fuente de la aplicación modelada, empleando para ello las funcionalidades implementadas en la clase *XFactoryControllerDCD.java*.

Paquete Xfactory.actions: Paquete que agrupa las clases asociadas a los servicios de captura de los elementos del diagrama de casos de uso del sistema y del diagrama entidad-relación, las cuales son *GeneratorAnalisysClass.java*, *GeneratorDesignClass.java* y *GeneratorSourceCode.java*. Incluye, además, clases que tienen como función la captura y creación de clases y las relaciones entre ellas a través de objetos de tipo *UseCaseDiagramaServices*, *EntityRelationDiagramServices*, *AnalisysClassDiagramServices* y *DesignClassDiagramServices*. Con el fin de llevar a cabo la generación de código fuente, este paquete incluye un conjunto de 15 plantillas creadas usando la tecnología *Apache Velocity 1.7*, mediante las cuales es posible la creación de clases Java Script correspondientes a Ext JS y PHP correspondientes a Symfony.

GeneratorAnalisysClass.java: Esta clase implementa las funcionalidades de la interfaz *com.vp.plugin.action.VPContextActionController* que se encargan de trabajar con los elementos del contexto del diagrama de casos de uso y mostrar el diálogo que permite generar las clases del análisis.

GeneratorDesignClass.java: Clase que implementa las funcionalidades de la interfaz *com.vp.plugin.action.VPContextActionController* que permite capturar los elementos del contexto de las clases del análisis para luego transformarlas en clases del diseño.

GeneratorSourceCode.java: Clase que tiene la responsabilidad de generar las clases que conforman el código fuente de la aplicación modelada, a través de la utilización de las plantillas de *Velocity* confeccionadas en el paquete *XFactory.actions* y de los métodos implementados en la clase *SourceCodeServices.java*.

Paquete Xfactory.dialog: Paquete que contiene las interfaces de usuario que son mostradas al analista y al desarrollador por la herramienta *Visual Paradigm for UML* cuando seleccionan las acciones contextuales “Generar Modelo del Análisis” y “Generar Código Fuente”.

DialogGeneratorAnalisysClass.java: Clase que muestra al analista una interfaz que reúne los campos referentes a los paquetes y casos de uso del sistema, así como la información asociada a los mismos, para que el analista seleccione los elementos que desee transformar al análisis.

DialogGeneratorSourceCode.java: Clase que muestra al desarrollador una interfaz que contiene un campo referente a la dirección de destino del código fuente generado, para que el desarrollador seleccione la dirección deseada y si desea generar un nuevo proyecto o incluir el código generado en un proyecto existente.

GUIVPPDeployer.jar: Es una herramienta de despliegue que facilita la integración de la extensión con *Visual Paradigm for UML*. Muestra una interfaz donde el desarrollador debe introducir los valores

iniciales “nombre de la extensión”, “dirección del proyecto de la extensión” y la “dirección donde será desplegado”.

3.3 Estándar de programación utilizado

Los estándares de programación reúnen un conjunto de técnicas de codificación sólidas y buenas prácticas de programación que propician que se genere un código de programación de alta calidad. Estas normas son de gran importancia para la calidad del producto de software. Algunas de las ventajas de utilizar estándares son:

- Facilitan el mantenimiento de una aplicación.
- Permiten que cualquier programador entienda y pueda mantener la aplicación.
- Mejoran la legibilidad del código, al mismo tiempo que permiten su compresión rápida.

En el desarrollo de la extensión para la herramienta *Visual Paradigm for UML* sobre la plataforma Java, se pusieron en práctica algunas de las convenciones recomendadas por *Sun Microsystems*. Entre ellas se pueden citar:

3.3.1 Organización de los ficheros

Serán evitados los ficheros de gran tamaño que contengan más de 1000 líneas de código, pues en ocasiones el tamaño excesivo provoca que la clase no encapsule un comportamiento claramente definido, albergando una gran cantidad de métodos que realizan tareas funcional o conceptualmente heterogéneas.

3.3.2 Tamaño y organización de las líneas de código

La longitud de línea no debe superar los 80 caracteres. En caso de que una expresión ocupe más de una línea, esta se podrá romper o dividir tras una coma o antes de un operador y la nueva línea debe estar alineada con el inicio de la expresión al mismo nivel que la línea anterior.

3.3.3 Declaraciones

Toda variable local tendrá que ser inicializada en el momento de su declaración, salvo que su valor inicial dependa de algún valor que tenga que ser calculado previamente. Las declaraciones deben situarse al principio de cada bloque principal en el que se utilicen y nunca en el momento de su uso.

En las declaraciones de los métodos, no debe incluirse ningún espacio entre el nombre del método y el paréntesis inicial del listado de parámetros. Los métodos se separarán entre sí mediante una línea en blanco.

3.3.4 Sentencias

El caracter inicio de bloque debe situarse al final de la línea que inicia el bloque. El caracter final de bloque debe situarse en una nueva línea tras la última línea del bloque y alineada con respecto al

primer caracter de dicho bloque. Todas las sentencias de un bloque deben encerrarse entre llaves, aunque el bloque conste de una única sentencia.

3.3.5 Espacios en blanco

Se utilizarán espacios en blanco entre una palabra clave y un paréntesis, tras cada coma en un listado de argumentos, para separar un operador binario de sus operandos, excepto en el caso del operador ("."), para separar las expresiones incluidas en la sentencia "for" y al realizar el moldeo o "casting" de clases.

3.3.6 Nomenclatura de identificadores

Paquetes: Los nombres de los paquetes se escribirán siempre en letras minúsculas para evitar que entren en conflicto con los nombres de clases e interfaces.

Clases e interfaces: Los nombres de clases deben ser sustantivos y deben tener la primera letra en mayúsculas. Si el nombre es compuesto, cada palabra componente deberá comenzar con mayúsculas. Debe evitarse el uso de acrónimos o abreviaturas. Toda interfaz se nombrará con el prefijo "I" para diferenciarla de la clase que la implementa (que tendrá el mismo nombre sin el prefijo "I").

Métodos: Los métodos deben ser verbos escritos en minúsculas. Cuando el método esté compuesto por varias palabras cada una de ellas tendrá la primera letra en mayúsculas.

Variables: Las variables se escribirán siempre en minúsculas. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúsculas. Las variables nunca podrán comenzar con el caracter "_" o "$". Los nombres de variables deben ser cortos y debe evitarse el uso de nombres de variables con un solo caracter, excepto para variables temporales.

Constantes: Todos los nombres de constantes tendrán que escribirse en mayúsculas. Cuando los nombres de constantes sean compuestos las palabras se separarán entre sí mediante el caracter de subrayado "_".

3.3.7 Buenas prácticas de programación

Visibilidad de atributos de instancia y de clase: Los atributos de instancia y de clase serán siempre privados, excepto cuando tengan que ser visibles en subclases herederas; en tales casos serán declarados como protegidos. El acceso a los atributos de una clase se realizará por medio de los métodos "get" y "set" correspondientes.

Referencias a miembros de una clase: Debe evitarse el uso de objetos para acceder a los miembros de una clase (atributos y métodos estáticos). Debe utilizarse en su lugar el nombre de la clase.

Asignación sobre variables: Se deben evitar las asignaciones de un mismo valor sobre múltiples variables en una misma sentencia, ya que dichas sentencias suelen ser difíciles de leer. No se deben utilizar asignaciones embebidas o anidadas.

Paréntesis: Se deben utilizar paréntesis en expresiones que incluyan distintos tipos de operadores para evitar problemas de precedencia de operadores.

3.4 Pruebas de Software

La prueba del software es un elemento crítico para garantizar la calidad del mismo, por lo que el objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Las pruebas son un proceso que se enfoca sobre la lógica interna y las funciones externas del software, las cuales consisten en la ejecución de un programa con la intención de descubrir un error. Una prueba tiene éxito si descubre un error no detectado hasta entonces. La etapa de pruebas implica acciones como[29]:

- Verificar la interacción de los componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.

Existen diferentes clasificaciones para las pruebas de software, cada una de ellas encaminada a probar un aspecto específico del sistema. Entre ellas se encuentran las pruebas de Caja Negra y pruebas de Caja Blanca. En el primero de los casos, las pruebas son realizadas sobre la interfaz del software, teniendo en cuenta los datos de entrada y estudiando si la respuesta del sistema es o no la esperada. El segundo caso utiliza la estructura de control del diseño procedimental para obtener los casos de prueba que serán aplicados.

3.4.1 Aplicación de las pruebas de software a la extensión

Con el objetivo de comprobar que la extensión funciona de forma correcta, son diseñadas e implementadas por el equipo de desarrollo pruebas correspondientes al primer nivel de prueba, es decir, pruebas de desarrollador. La técnica de prueba que se aplicará es la de pruebas de funcionalidad, las cuales fijan su atención en la validación de las funciones, métodos, servicios y casos de uso. Durante la aplicación de esta técnica se analiza cada funcionalidad implementada para verificar que se cumplan todos los requisitos establecidos y de esta forma satisfacer las necesidades planteadas. Los objetivos de este tipo de pruebas contemplan la navegación, entrada de datos, procesamiento y obtención de resultados, enfocándose en los requisitos funcionales y casos de uso.

De acuerdo con las características y objetivos a los que están encaminados, se decidió aplicar a la extensión pruebas de Caja Blanca para identificar los casos de prueba que serán diseñados y pruebas de Caja Negra para, dado un conjunto de condiciones de entrada, lograr que se ejerciten todos los

requisitos funcionales de la extensión. A continuación se describe y ejemplifica el proceso de pruebas realizado.

3.4.2 Aplicación de pruebas de Caja Blanca

Entre los métodos de prueba de Caja Blanca se encuentra el del camino simple, el cual se aplica a fragmentos del código fuente de la aplicación. Por ser una de las más significativas de la extensión, se determinó aplicar este método a la funcionalidad que genera los diagramas de clases pertenecientes al modelo del análisis. Para la función "GenerarModelos", perteneciente a la clase *DialogGeneratorAnalisysClass.java*, fueron identificados y enumerados los bloques de ejecución como se muestra a continuación:

```
private void GenerarModelos() {
    DefaultMutableTreeNode elem = (DefaultMutableTreeNode) jTree1.getLastSelectedPathComponent (); 1
    if (elem == null) { 2
        JOptionPane.showMessageDialog(null, "Debes elegir el elemento a generar"); 3
    } else if (elem.isRoot()) { 4
        for (int i = 0; i < controler.getArrayUseCasesContext().length; i++) { 5
            controler.transformModelAnalisys(controler.getArrayUseCasesContext()[i]); 6
        }
        this.dialog.close(); 7
    } else if (elem.isLeaf()) { 8
        for (int i = 0; i < controler.getArrayUseCasesContext().length; i++) { 9
            IUseCase crudChoosen = controler.getArrayUseCasesContext()[i]; 10
            if (crudChoosen.getName().equals(elem.getUserObject().toString())) { 11
                controler.transformModelAnalisys(crudChoosen); 12
            }
        }
        this.dialog.close(); 13
    } else { 14
        for (int i = 0; i < controler.getArrayModuleContext().length; i++) { 15
            IPackage moduleChoosen = controler.getArrayModuleContext()[i]; 16
            if (moduleChoosen.getName().equals(elem.getUserObject().toString())) { 17
                for (int j = 0; j < controler.getArrayUseCasesContext().length; j++) { 18
```

```
            IUseCase iUseCase = controler.getArrayUseCasesContext()[j];  19
            if (iUseCase.getParent().getName().equals(elem.getUserObject().toString())) {  20
              controler.transformModelAnalisys(iUseCase);  21
            }
          }
        }
      }
      this.dialog.close();  22
    }
  }
```

De ese modo se obtuvieron 22 bloques, se dibujó el grafo de flujo asociado y se determinó el camino básico. Como se muestra en el grafo de flujo (ver Figura 13), los nodos resaltados corresponden a los nodos predicados y las aristas indican los posibles caminos a seguir a partir del nodo correspondiente. Partiendo del camino básico determinado, fue aplicado uno de los tres métodos para calcular la complejidad ciclomática, específicamente el método V (G) = A – N + 2. Se obtuvieron 30 aristas y 22 nodos, quedando la fórmula de la siguiente forma: V (G) = 30 – 22 + 2. Por lo tanto, la complejidad ciclomática tiene un valor de 10, lo cual significa que existen 10 posibles casos de ejecución para la función, información que fue utilizada a la hora de diseñar los casos de prueba de Caja Negra.

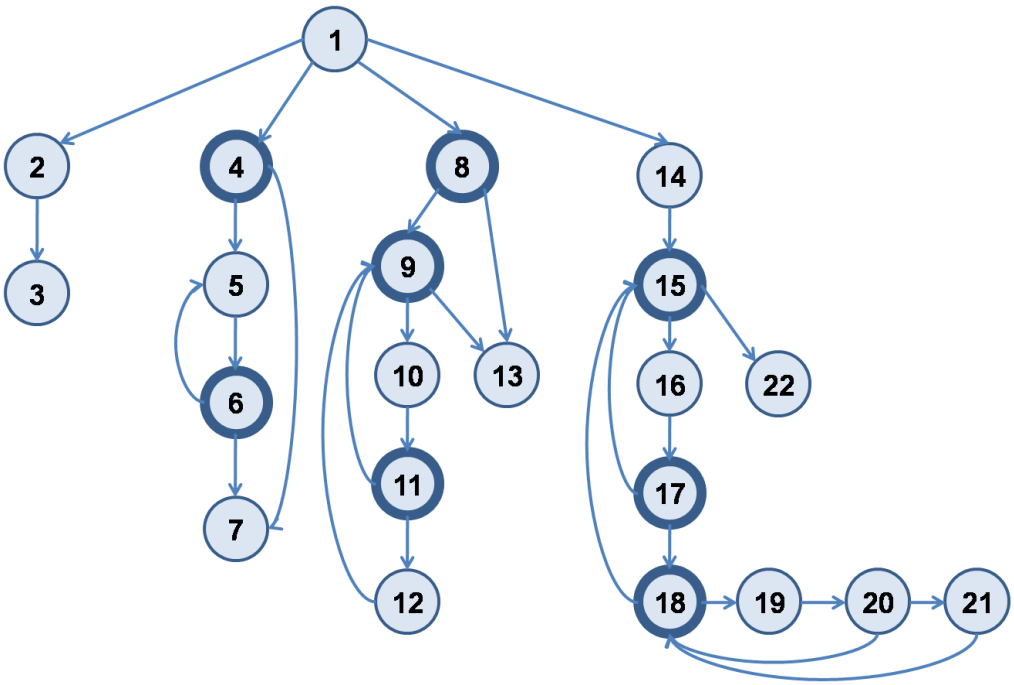


Figura 13: Grafo de flujo que muestra el camino básico de la función "GenerarModelos"

3.4.3 Aplicación de pruebas de Caja Negra

Las pruebas de Caja Negra fueron aplicadas mediante casos de prueba, los cuales están compuestos por un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollados para cumplir un objetivo en particular o una función esperada. Dichas pruebas se encuentran reflejadas en los artefactos DCP_CU_Generar Modelo del Análisis y DCP_CU_Generar Código Fuente (Ver Expediente de Proyecto). A continuación se presentan los casos de prueba correspondientes al caso de uso “Generar Modelo del Análisis”, específicamente para la sección “Realizar Caso de Uso”, aplicados en la primera iteración de pruebas. Dichas pruebas se realizan a través de una matriz de datos, donde:

V: indica válido

I: indica inválido

NA: indica que no es necesario proporcionar un valor del dato, ya que es irrelevante.

En la siguiente tabla se muestran las variables V1,…, V7 que representan valores de entrada de datos para los casos de prueba aplicados al caso de uso “Generar Modelo del Análisis”.

Descripción de las variables

Tabla 7: Descripción de las variables correspondientes al caso de prueba para el caso de uso “Generar Modelo del Análisis”

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
Generar Modelo del Análisis				
V1	Paquetes y casos de uso del sistema.	árbol	no	Árbol que contiene los paquetes y casos de uso del sistema. Inicialmente no se encuentra seleccionado. Debe ser seleccionado el sistema, un paquete o un caso de uso.
V2	Pref.: Form*	campo de texto	no	Campo que solo admite caracteres. Inicialmente contiene el prefijo Form por defecto, aunque puede ser modificado.
V3	Pref.: Registro	campo de texto	si	Campo que solo admite caracteres. Inicialmente se encuentra vacío, aunque puede ser modificado.

V4	Pref.: Dialogo*	campo de texto	no	Campo que solo admite caracteres. Inicialmente contiene el prefijo Detail por defecto, aunque puede ser modificado.
V5	Pref.: Colección*	campo de texto	no	Campo que solo admite caracteres. Inicialmente contiene el prefijo Collection por defecto, aunque puede ser modificado.
V6	Pref.: Maestro*	campo de texto	no	Campo que solo admite caracteres. Inicialmente contiene el prefijo Master por defecto, aunque puede ser modificado.
V7	Datos del elemento seleccionado.	tabla	no	Tabla que solo admite caracteres como valores de sus celdas. Inicialmente contiene los datos correspondientes al elemento seleccionado, aunque puede ser modificado.

A continuación se muestra el caso de prueba aplicado al caso de uso “Generar Modelo del Análisis”, específicamente a la sección “Realizar Caso de Uso”:

Tabla 8: Caso de prueba aplicado al caso de uso "Generar Modelo del Análisis"

Escenario	Descripción	V1	V2	V3	V4	V5	V6	V7	Respuesta del sistema	Flujo central
SC1.Realizar Caso de Uso, con datos correctos.	En este escenario se genera el modelo del análisis a partir del diagrama entidad-relación y del diagrama de casos de uso del sistema.	V: Se selecciona un caso de uso. (Gestionar Usuario)	V: String (Form)	V: -[ ]	V: String (Detail)	V: String (Collection)	V: String (Master)	V: String	La salida será los nuevos diagramas de clases del análisis generados a partir del diagrama entidad-relación y del diagrama de casos de uso del sistema, con todas las entidades transformadas a clases, los campos en atributos y todas las operaciones convertidas en métodos.	1. Desde el contexto de trabajo el analista debe pulsar la opción "Generar Modelo del Análisis". 2. En la interfaz mostrada por la extensión debe seleccionar un caso de uso. 3. Debe introducir las variables de entrada según correspondan. 4. Debe pulsar el botón "Generar"; acto seguido serán generados los diagramas de
SC1. Realizar Caso de Uso, con datos incorrectos.	En este escenario se genera el modelo del análisis a partir del diagrama entidad-relación y del	I: Se selecciona un caso de uso (Gestionar Usuario) y se dejan campos vacíos.	V: String (Form)	V: -[ ]	V: -[ ]	V: String (Collection)	V: String (Master)	V: String	El sistema debe mostrar una ventana alertando al analista mediante el mensaje "Debe llenar todos los campos obligatorios" y no genera los diagramas de clases del análisis.	

	diagrama de casos de uso del sistema.									clases del análisis.
		I: No se selecciona ningún caso de uso.	V: -[ ]	V: -[ ]	V: -[ ]	V: -[ ]	V: -[ ]	V: -[ ]	El sistema debe mostrar una ventana alertando al analista mediante el mensaje “Debe elegir un elemento a generar” y no genera los diagramas de clases del análisis.	
		I: Se selecciona un caso de uso (Gestionar Usuario) y se introducen caracteres especiales	V: String (Form)	V: -[ ]	V: String (De@ail)	V: String (Collection)	V: String (Mast”er)	V: String	El sistema debe mostrar una ventana alertando al analista mediante el mensaje “No debe introducir caracteres especiales en los campos” y no genera los diagramas de clases del análisis.	

Para validar el correcto funcionamiento de la extensión fueron realizadas tres iteraciones de pruebas. Cada dificultad detectada y resuelta se encuentra reflejada en el artefacto No Conformidades (Ver Expediente de Proyecto). En la siguiente tabla se muestra un resumen general de todas las dificultades encontradas por iteraciones en las pruebas realizadas, donde:

PD: Pendiente

RA: Resuelta

Tabla 9: Resumen de los resultados de las pruebas aplicadas

Fecha	Versión	Caso de prueba	Cant. No conformidad	Cant. No conformidades PD	Cant. No conformidades RA
10/04/2012	1.0	CU Generar Modelo del Análisis.	2	-	2
19/05/2012	1.2	CU Generar Modelo del Análisis. CU Generar Código Fuente.	2 1	- -	2 1
7/06/2012	1.2	CU Generar Modelo del Análisis. CU Generar Código Fuente.	0 0	- -	- -

Luego de realizar las pruebas de Caja Negra en tres iteraciones de prueba, mediante los casos de prueba asociados a cada caso de uso, se comprobó que la extensión da cumplimiento de forma correcta a todos los requisitos funcionales definidos. Fueron validadas las entradas de datos en las interfaces de usuario, para que la generción de los diagramas del análisis y del código fuente se realice solo con datos correctos. Además, se validó que la realización de los diagramas de clases del diseño tenga como entradas los diagramas de clases del análisis y que los diagramas de componentes y la generación de código fuente se realice solamente a partir de diagramas de clases del diseño. Se comprobó que la aplicación construida a partir de la generación de código fuente funcione de forma correcta.

Conclusiones parciales

La descripción de las consideraciones que se deben tener en cuenta para la implementación de extensiones para la herramienta *Visual Paradigm for UML* permitió un mayor entendimiento, facilitando

el proceso de desarrollo e integración de la extensión. Luego de analizar las relaciones que existen entre los componentes de hardware y software en el sistema, se concluyó que no es necesario confeccionar el diagrama de despliegue debido a que todos los elementos de la extensión están contenidos en un único nodo físico.

A partir de un análisis realizado acerca de los componentes fundamentales asociados al diseño de clases propuesto para la construcción de la extensión, fueron identificados y descritos 24 componentes que conforman el diagrama de componentes del sistema. Para la implementación de la extensión propuesta se consideró necesario aplicar algunos estándares de programación para garantizar que la codificación se realice correctamente. La realización de pruebas de Caja Negra y Caja Blanca permitió validar que los requisitos funcionales de la extensión fueron implementados correctamente.

CONCLUSIONES

Como resultado del presente trabajo de diploma y dando cumplimiento a los objetivos propuestos, se puede establecer como conclusiones las siguientes:

- A partir del estudio de los elementos teóricos para el desarrollo de aplicaciones de gestión de información, se determinó que las metodologías MDE y MDD fueron las más idóneas para ser aplicadas en el desarrollo de la solución. La puesta en práctica de las mismas permitió que la generación de código y las transformaciones entre diagramas se realizara de forma automática, partiendo de los modelos contenidos en los diagramas.

- La definición y especificación de un nuevo perfil UML para la implementación de la extensión permitió adaptar los modelos provistos por UML al dominio del TPS modelado en la heramienta.

- La aplicación de los mecanismos de análisis, de diseño y de implementación constituyó un factor clave para realizar las transformaciones de diagramas hacia las siguientes etapas de forma correcta, garantizando la trazabilidad.

- Teniendo en cuenta las exigencias y necesidades del centro DATEC relacionadas con la construcción de TPS, fueron identificados e implementados los requisitos de la extensión, obteniéndose como resultado ocho requisitos funcionales que fueron agrupados en cuatro casos de uso. La puesta en práctica del patrón arquitectónico Modelo-Vista-Controlador permitió la obtención de una arquitectura robusta para la extensión y la aplicación de los patrones GOF y GRASP permitió el diseño adecuado y la correcta asignación de responsabilidades a cada una de las clases implementadas. De ese modo se obtuvo una extensión para *Visual Paradigm for UML* capaz de realizar transformaciones sucesivas y automáticas entre los diagramas de las etapas de análisis, diseño e implementación, además de la generación del código fuente de la aplicación modelada.

- La validación de las funcionalidades implementadas, a través de pruebas funcionales, utilizando los métodos de Caja Negra y de Caja Blanca, permitió detectar errores en el comportamiento de la extensión. Los principales errores detectados durante las tres iteraciones de pruebas aplicadas estuvieron relacionados con la validación de los datos de entrada de las interfaces y con comportamientos inadecuados de las funcionalidades implementadas. Finalmente, fueron resueltas todas las no conformidades detectadas, obteniéndose la correcta implementación de cada requisito funcional de la extensión.

RECOMENDACIONES

Se recomienda para posteriores versiones de la extensión, implementar las siguientes funcionalidades:

- Generar los diagramas de clases del diseño utilizando estereotipos web en caso de que el desarrollador así lo desee.
- Generar los diagramas correspondientes al modelo del análisis, modelo del diseño y modelo de implementación, así como generar el código fuente de la aplicación no solo a partir de CRUD completo, sino también a partir de CRUD parcial.
- Realizar la ingeniería inversa de aplicaciones de gestión de información implementadas en Symfony y Ext JS.

REFERENCIAS BIBLIOGRÁFICAS

1. Ruz, R.C., *Decreto- Ley No. 281 "Del Sistema de Información del Gobierno". Gaceta Oficial de la República de Cuba,* 2011, No 10: p. 29.
2. Kenneth E Kendall, J.E.K.y.A.N.R., *Análisis y diseño de sistemas. Pearson Educación*
3. Garzás, J. *Los sistemas de información: importancia, fundamentos, calidad y gestión estratégica de las tecnologías de la información.* [en línea].[Consultado el: 29 de noviembre del 2011]. Disponible en: http://aprendeenlinea.udea.edu.co/lms/moodle/file.php/516/MODULO_1/ApuntesSistemasInformacion.pdf
4. Antonio Cicchetti, o., *Automating Co-evolution in Model-Driven Engineering.*: Italia.
5. José Manuel Pérez, F.R., Mario Piattini, *Model Driven Engineering Aplicado a Business Process Management.* 2007.
6. *Desarrollo Dirigido por Modelos y generación automática de código.* [en línea].[Consultado el: 5 de diciembre del 2011]. Disponible en: http://www.pros.upv.es/index.php/es/lineas/69-lineaddm
7. *El desarrollo dirigido por modelos (MDD). Modelos y Proyectos.* [en línea]. [Consultado el: 5 de diciembre del 2011]. Disponible en: http://www.modelosyproyectos.com/2010/01/25/el-desarrollo-dirigido-por-modelos-mdd/
8. L. Cuaderno, o., *Herramientas de soporte al proceso de desarrollo dirigido por modelos y su implementación con DSL Tools.*: Argentina.
9. *Modelado de Sistemas con UML.* [en línea]. [Consultado el: 10 de diciembre del 2011]. Disponible en: http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf
10. Blanco., Y.G., *Perfil de UML para los proyectos de la línea Soluciones Integrales. Tesis de Diploma de Ingeniería en Ciencias Informáticas.* Universidad de las Ciencias Informáticas.: La Habana, Cuba, 2011.
11. Haywood, D., *Domain-Driven Design Using Naked Objects.* 2009: Texas.
12. ARTech, *GeneXus. Visión general.* 2003.
13. *OpenUp*. Ecured. [en línea]. [Consultado el: 28 de noviembre del 2011]. Disponible en: http://www.ecured.cu/index.php/OpenUp
14. *Java.* Ecured. [en línea]. [Consultado el: 29 de noviembre del 2011]. Disponible en: http://www.ecured.cu/index.php/Java
15. *Visual Paradigm*. [en línea]. [Consultado el: 30 de noviembre del 2011]. Disponible en: http://www.visual-paradigm.com/

16. *NetBeans*. [en línea].[*Consultado el: 14 de marzo del 2012]*. Disponible en: http://netbeans.org/index_es.html
17. *The Apache Velocity Project*. [en línea]. [Consultado el: 14 de marzo del 2012]. Disponible en: http://velocity.apache.org
18. *Flujo de Trabajo Requerimiento*. Ecured. [en línea] .[Consultado el: 22 de febrero del 2012]. Disponible en: http://www.ecured.cu/index.php/Flujo_de_Trabajo_Requerimiento#Requerimientos_Funcionales
19. *Especificación de Requerimientos. Diseño de Base de Datos*. [en línea]. [Consultado el: 19 de febrero del 2012]. Disponible en: http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf
20. *El Modelo de Caso de Uso.* [en línea]. [Consultado el: 19 de febrero del 2012]. Disponible en: http://www.sparxsystems.com.ar/resources/tutorial/use_case_model.html
21. Rational Unified Process, *Ayuda del Proceso Unificado de Rational*. 2005 [Consultado el: 26 de abril de 2012]
22. UPEDU, *Unified Process for EDUcation: Artifacts.* [en línea]. [Consultado el: 27 de abril de 2012]. Disponible en: http://www.upedu.org/process/artifact/ar_desmd.htm
23. Universidad Politécnica de Madrid, *Patrones del "Grang of Four"*. [en línea]. [Consultado el: 27 de abril de 2012]. Disponible en: http://is.ls.fi.upm.es/docencia/proyecto/docs/patrones_gof.pdf
24. *Patrones de Asignación de Responsabilidades.* [en línea]. [Consultado el: 28 de abril de 2012]. Disponible en: http://www.ecured.cu/index.php/Patrones_de_Asignaci%C3%B3n_de_Responsabilidades
25. James Rumbaugh, I.J., Grady Booch., *El lenguaje de Modelado.* 1998.
26. Yenisleydis Ledesma Rodríguez, Y.B.R., *Extensión de la herramienta "Visual Paradigm for UML" para el soporte al Desarrollo Dirigido por Modelos con Ext JS. Tesis de Diploma de Ingeniería en Ciencias Informáticas.* Universidad de las Ciencias Informáticas.: La Habana, Cuba, 2011.
27. Ivar Jacobson, G.B., James Rumbaugh, *El Proceso Unificado de Desarrollo de Software.*
28. Daniele, M. *Teoría 11: El Arte de Modelar UML*. 2007. [en línea]. [Consultado el: 28 de abril de 2012]. Disponible en: http://es.scribd.com/doc/57150601/UML-DiagramaComponentes
29. *TiposPruebaSoftware*. *[en línea].* [Consultado el: 23 de marzo de 2012]. Disponible en: http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf

BIBLIOGRAFÍA

1. Ruz, R.C., *Decreto- Ley No. 281 "Del Sistema de Información del Gobierno". Gaceta Oficial de la República de Cuba,* 2011, No 10: p. 29.
2. Kenneth E Kendall, J.E.K.y.A.N.R., *Análisis y diseño de sistemas. Pearson Educación*
3. Garzás, J. *Los sistemas de información: importancia, fundamentos, calidad y gestión estratégica de las tecnologías de la información.* [en línea]. [Consultado el: 29 de noviembre del 2011]. Disponible en: http://aprendeenlinea.udea.edu.co/lms/moodle/file.php/516/MODULO_1/ApuntesSistemasInformacion.pdf
4. Antonio Cicchetti, o., *Automating Co-evolution in Model-Driven Engineering.*: Italia.
5. José Manuel Pérez, F.R., Mario Piattini, *Model Driven Engineering Aplicado a Business Process Management.* 2007.
6. *Desarrollo Dirigido por Modelos y generación automática de código.* [en línea].[Consultado el: 5 de diciembre del 2011]. Disponible en: http://www.pros.upv.es/index.php/es/lineas/69-lineaddm
7. *El desarrollo dirigido por modelos (MDD). Modelos y Proyectos.* [en línea]. [Consultado el: 5 de diciembre del 2011]. Disponible en: http://www.modelosyproyectos.com/2010/01/25/el-desarrollo-dirigido-por-modelos-mdd/
8. L. Cuaderno, o., *Herramientas de soporte al proceso de desarrollo dirigido por modelos y su implementación con DSL Tools.*: Argentina.
9. *Modelado de Sistemas con UML.* [en línea]. [Consultado el: 10 de diciembre del 2011]. Disponible en: http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf
10. Blanco., Y.G., *Perfil de UML para los proyectos de la línea Soluciones Integrales. Tesis de Diploma de Ingeniería en Ciencias Informáticas.* Universidad de las Ciencias Informáticas.: La Habana, Cuba, 2011.
11. Haywood, D., *Domain-Driven Design Using Naked Objects.* 2009: Texas.
12. ARTech, *GeneXus. Visión general.* 2003.
13. *OpenUp*. Ecured. [en línea]. [Consultado el: 28 de noviembre del 2011]. Disponible en: http://www.ecured.cu/index.php/OpenUp
14. *Java.* Ecured. [en línea]. [Consultado el: 29 de noviembre del 2011]. Disponible en: http://www.ecured.cu/index.php/Java
15. *Visual Paradigm*. [en línea]. [Consultado el: 30 de noviembre del 2011]. Disponible en: http://www.visual-paradigm.com/

16. *NetBeans*. [en línea].[Consultado el: 14 de marzo del 2012]. Disponible en: http://netbeans.org/index_es.html
17. *The Apache Velocity Project*. [en línea]. [Consultado el: 14 de marzo del 2012]. Disponible en: http://velocity.apache.org
18. *Flujo de Trabajo Requerimiento*. Ecured. [en línea] .[Consultado el: 22 de febrero del 2012]. Disponible en: http://www.ecured.cu/index.php/Flujo_de_Trabajo_Requerimiento#Requerimientos_Funcionales
19. *Especificación de Requerimientos. Diseño de Base de Datos.* [en línea]. [Consultado el: 19 de febrero del 2012]. Disponible en: http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf
20. *El Modelo de Caso de Uso.* [en línea]. [Consultado el: 19 de febrero del 2012]. Disponible en: http://www.sparxsystems.com.ar/resources/tutorial/use_case_model.html
21. Rational Unified Process, *Ayuda del Proceso Unificado de Rational*. 2005 [Consultado el: 26 de abril de 2012]
22. UPEDU, *Unified Process for EDUcation: Artifacts.* [en línea]. [Consultado el: 27 de abril de 2012]. Disponible en: http://www.upedu.org/process/artifact/ar_desmd.htm
23. Universidad Politécnica de Madrid, *Patrones del "Grang of Four"*. [Consultado el 27 de abril de 2012]. Disponible en: http://is.ls.fi.upm.es/docencia/proyecto/docs/patrones_gof.pdf
24. *Patrones de Asignación de Responsabilidades.* [en línea]. [Consultado el: 28 de abril de 2012]. Disponible en: http://www.ecured.cu/index.php/Patrones_de_Asignaci%C3%B3n_de_Responsabilidades
25. James Rumbaugh, I.J., Grady Booch., *El lenguaje de Modelado.* 1998.
26. Yenisleydis Ledesma Rodríguez, Y.B.R., *Extensión de la herramienta "Visual Paradigm for UML" para el soporte al Desarrollo Dirigido por Modelos con Ext JS. Tesis de Diploma de Ingeniería en Ciencias Informáticas.* Universidad de las Ciencias Informáticas.: La Habana, Cuba, 2011.
27. Ivar Jacobson, G.B., James Rumbaugh, *El Proceso Unificado de Desarrollo de Software.*
28. Daniele, M. *Teoría 11: El Arte de Modelar UML*. 2007. [en línea]. [Consultado el: 28 de abril de 2012]. Disponible en: http://es.scribd.com/doc/57150601/UML-DiagramaComponentes
29. *TiposPruebaSoftware*. [en línea]. [Consultado el: 23 de marzo de 2012]. Disponible en: http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf
30. Franck Fleurey, J.S., Benoit Baudry. *Validation in Model-Driven Engineering: Testing Model Transformations.* **Volume**
31. Jean-Marie Favre, T.N. *Towards a Megamodel to Model Software Evolution Through Transformations.* **Volume**

32. *Symfony*. [en línea]. [Consultado el: 19 de noviembre del 2011]. Disponible en: http://www.librosweb.es/symfony/capitulo1.html
33. Cañavate., A.M. *Sistemas de información en las empresas.* [en línea]. [Consultado el: 16 de noviembre del 2011]. Disponible en: http://www.upf.edu/hipertextnet/
34. *SELECTING A DEVELOPMENT APPROACH.* [en línea]. [Consultado el: 2 de diciembre del 2011]. Disponible en: https://www.cms.gov/systemlifecycleframework/downloads/selectingdevelopmentapproach.pdf
35. R. E. Johnson, V.F.R., *Reusing Object-Oriented Designs*, in *Department of Computer Science*. 1991, University of Illinois: Urbana. p. 40.
36. Ortiz, H.K. *Representacion del Modelo de Objetos de Dominio*. 2009. [en línea]. [Consultado el: 17 de febrero del 2012]. Disponible en: http://www.eumed.net/libros/2009c/583/Representacion%20del%20Modelo%20de%20Objetos%20de%20Dominio.htm
37. *Pruebas de caja negra*. [en línea]. [Consultado el: 5 de mayo del 2012]. Disponible en: http://www.ecured.cu/index.php/Pruebas_de_caja_negra
38. *Pruebas de Caja Blanca*. [en línea]. [Consultado el: 10 de mayo del 2012]. Disponible en: http://www.ecured.cu/index.php/Pruebas_de_caja_blanca
39. Calderón Rivero, P.S., Vence Naranjo. , *Monografía. Sistema información económico-financiero- contable y la auditoría a los sistema inventario.* 2009.
40. *Modelo de Dominio.* 2008. [en línea]. [Consultado el: 17 de febrero del 2012]. Disponible en: http://synergix.wordpress.com/2008/07/10/modelo-de-dominio/
41. Bézivin., J. *Model Driven Engineering: An Emerging Technical Space.* **Volume**
42. Abel Avram, F.M., *Domain-Driven Design Quickly.*, ed. I. 978-1-4116-0925-9. 2006, United States of America.
43. *Domain-Driven Design Community*. [en línea]. [Consultado el: 28 de noviembre del 2011]. Disponible en: http://www.domaindrivendesign.org/
44. Vega, M. *Casos de uso UML*. 2010. Granada.
45. *Caso de uso*. [en línea]. [Consultado el: 29 de noviembre del 2011]. Disponible en: http://www.ecured.cu/index.php/Caso_de_uso
46. *Ayuda de OpenUP Version 1.5.0.1. OpenUP*. [en línea]. [Consultado el: 25 de noviembre del 2011]. Disponible en: http://epf.eclipse.org/wikis/openup

ANEXOS

Anexo 1. Aplicación del patrón Singleton o Solitario en la extensión

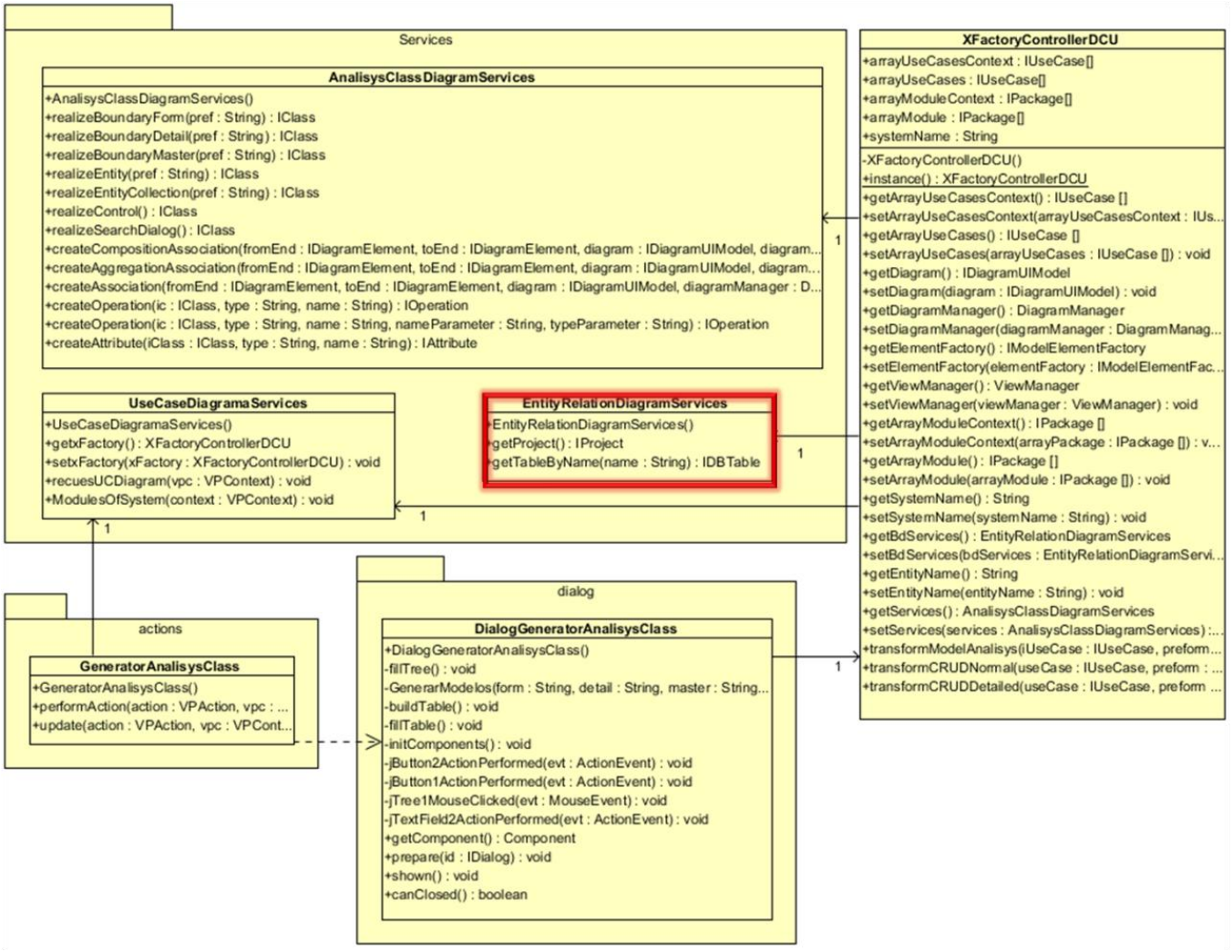


Figura 14: Ejemplo de clase donde es aplicado el patrón Solitario